Computational Protein Design with Deep Learning and Automated Reasoning

Marianne Defresne

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse

---

Présentée et soutenue par

Marianne DEFRESNE

Le 30 novembre 2023

Le design de protéines par apprentissage profond et raisonnement automatique

---

Ecole doctorale : **SEVAB - Sciences Ecologiques, Vétérinaires, Agronomiques et Bioingenieries**

Spécialité : **Ingénieries microbienne et enzymatique**

Unité de recherche :
TBI - Toulouse Biotechnology Institute, Bio & Chemical Engineering

Thèse dirigée par
Sophie BARBE et Thomas SCHIEX

Jury

M. Jean-Christophe GELLY, Rapporteur
M. Tias GUNS, Rapporteur
Mme Céline HUDELOT, Présidente du Jury
M. Sergey OVCHINNIKOV, Examinateur
Mme Sophie BARBE, Directrice de thèse
M. Thomas SCHIEX, Co-directeur de thèse

À mes grand-mères

Acknowledgments

Cette thèse est le résultat de trois ans de travail, qui a été rendu possible grâce au soutien de nombreuses personnes, tant scientifiquement que personnellement. Je tenais donc à commencer ce manuscrit en les remerciant.

Je remercie d'abord et avant tout mes directeurs de thèse, Thomas Schiex et Sophie Barbe. Merci de m'avoir confié ce sujet, puis de m'avoir accompagnée tout en m'offrant l'autonomie dont j'avais besoin pour l'explorer. Travailler avec vous a été un privilège, et j'ai apprécié chacun des jours que j'ai passé en thèse.

Je remercie ensuite chacun des membres de mon jury. Tout d'abord, merci à mes rapporteurs Jean-Christophe Gelly et Tias Guns d'avoir accepter de relire et évaluer ce manuscrit. Merci à mes examinateurs Céline Hudelot et Sergey Ovchinnikov pour leur expertise et leur intérêt. Je remercie également les membres de mon comité de suivi, Sergei Grudinin et Nicolas Dobigeon pour avoir suivi l'évolution de mon travail. Enfin, merci à l'EUR BioEco et à ANITI d'avoir financé cette thèse.

Durant ma thèse, j'ai eu la chance de travailler dans deux équipes, chacune composée de collègues formidables. Je remercie donc toute l'EAD1 de TBI, et particulièrement le Modélo gang: Romain, Younes, Marc, Charlotte, Delphine, Bessam, Agata, Nicolas, Maud, Dhoha, Elin, Léopold, Carla, Isabelle et Jérémy. Merci pour tous ces moments partagés, au labo et en dehors, et pour votre enthousiasme à aider une informaticienne à comprendre quelque chose aux protéines. Je remercie également chacun des membres de MIAT, et particulièrement la chaire ANITI: Simon, Pierre, Valentin, David. Merci également à tous ceux avec qui j'ai partagé un bureau, un tarot ou une rando. Enfin, je remercie mon stagiaire Romain pour son enthousiasme et ses idées. Merci à tous pour votre accueil !

Merci à ma colocataire Paloma, qui a suivi de près chaque moment de ma thèse, du concours de l'école doctorale à la soutenance; merci aussi aux

autres latinistes, Karen, Claire et Pauline d'être des amies géniales depuis de nombreuses années. Je tiens également à remercier les télécomiens, Romain, Jacques, Léna, Tristan, Inasse, Constance et Florent. Vous voir tous à ma soutenance m'a vraiment fait chaud au coeur.

Merci à Samuel pour ces dernies mois, et, je l'espère, ceux à venir. Et surtout merci à ma famille sur qui je peux toujours compter. Merci à ma grand-mère et à mon oncle, toujours prêt à m'accueillir pour me resourcer à la campagne. Enfin et surtout, merci à mon père et ma belle-mère, à ma grande soeur Virginie et mes petites soeurs Lucie et Laura.

Contents

Introduction

Proteins are complex molecules that are essential to life. Indeed, they are in charge of many processes within cells: they provide structure, catalyze reactions, transmit signals and much more. This wide variety of functions can be repurposed for many applications in biotechnologies, medicine, green chemistry, etc. If billion of years of evolution have adapted proteins to perform enhanced or new functions for biological needs, industrial applications present specific conditions for which natural proteins may not be suited. Computational Protein Design (CPD) aims at finding new proteins with desired properties or functions.

Proteins are composed of a succession of small molecules called amino acids. Most proteins fold into a 3D shape, determined by the physico-chemical properties of their amino acids. Since the function of a protein is tightly linked to its 3D structure, CPD consists in crafting a structure hosting the targeted characteristics, then finding finding a sequence that folds into the target structure. We focus on sequence design. Sequence search can be reformulated as a discrete reasoning problem seeking to minimize an energy-score capturing interactions within the protein. Existing scoring functions are based on physics approximation and/or statistics and their quality may limit practical design.

In this work, we aim to capture more finely the sequence-structure relationships of natural proteins by deep learning a new score function. This score function will be optimized with existing discrete reasoning tools to design new proteins. For practical applications, it can optionally be combined with additional constraints or knowledge to better guide the design toward a protein sequence satisfying all the design requirements.

Our main objective faces two challenges. First, protein structures define challenging data for learning, requiring dedicated neural architectures. Second, we aim to develop a pipeline combining learning and reasoning. Building such a hybrid compound is one of the open challenges of Artificial Intelligence. In a divide-and-conquer spirit, we tackle both difficulties separately.

Organisation of the manuscript

This manuscript is organized in 5 chapters.

In Chapter 1, we introduce all the notions that will be used in this interdisciplinary work: protein basics, Deep Learning and the automated reasoning compound we use: Graphical Models. In Chapter 2, we review existing approaches for Computational Protein Design, with a particular emphasis on the most recent methods based on Deep Learning (DL). Discussing the strengths and weaknesses of the 2 main kinds of approaches, DL-based and energy-based, will lead us to the exact formulation of our main goal which aims to take the best of both worlds.

Chapter 3 will digress slightly from proteins to focus on hybrid AI. We take a brief overview of existing approaches to place our problem in the right framework, so-called Decision-Focused Learning. We focus on a toy problem with interesting parallels to CPD, learning how to play the game of Sudoku. We introduce a new loss for scalable coupling of Deep Learning with discrete reasoning.

We are back to proteins in Chapter 4 where we detail and discuss the protein representation we chose, together with the architecture we use to learn our score function for CPD. We optimize it to design proteins in Chapter 5. Extensive *in silico* (*i.e.*, computational) validation is performed before applying our score function to practical design projects.

Thesis contributions

A review of the Deep Learning methods for CPD. The literature about DL on protein data is recent but very rich. We classified existing DL approaches for design based on how the protein was represented. If this work dates back to 2021, it is extended to include more recent approaches in Chapter 2.

Marianne Defresne, Sophie Barbe, and Thomas Schiex (2021). *Protein Design with Deep Learning.* International Journal of Molecular Sciences 22.21. issn: 1422-0067

A loss for scalably coupling Deep Learning and discrete reasoning. Our formulation of CPD requires a hybrid pipeline that scales to large instances as proteins contain hundreds of amino acids or more. We also want

to benefit from exact solving at inference. Since no existing methods offered both advantages, we developed a different approach based on a dedicated loss.

Marianne Defresne, Sophie Barbe, and Thomas Schiex (2023). *Scalable Coupling of Deep Learning with Logical Reasoning.* Thirty-second International Joint Conference on Artificial Intelligence, IJCAI'2023.

A deep learned scoring function for CPD. We introduce Effie, our deep-learned score function for protein design. We show through *in silico* and experimental validation that optimizing Effie results in good-quality protein sequences. Finally, we provide evidence of some advantages of optimizing a score function over pure DL approaches.

Marianne Defresne et al. *Computational Protein Design with Hybrid Artificial Intelligence.* In preparation.

Delphine Dessaux, Samuel Buchet, **Marianne Defresne**, Simon de Givry, Thomas Schiex, Sophie Barbe. *Negative design for specific interfaces in protein assemblies.* In preparation.

Oral presentations

Marianne Defresne, Thomas Schiex, Sophie Barbe. 2023. *Scalable Coupling of Deep Learning with Logical Reasoning.* IJCAI 2023, Macao, S.A.R.

Marianne Defresne, Thomas Schiex, Sophie Barbe. 2023. *Scalable Coupling of Deep Learning with Logical Reasoning.* France@International at PFIA 2023, Strasbourg, France.

Marianne Defresne, Thomas Schiex, Sophie Barbe. 2023. *Computational Protein Design with Artificial Intelligence.* GGMM 2023, Toulouse, France.

Marianne Defresne, Thomas Schiex, Sophie Barbe. 2023. *Learning to reason: Embedding the solver or not Embedding the solver?.* Constraint Programming and Machine Learning Bridge at AAAI 2023, Washington DC, USA.

Marianne Defresne, Thomas Schiex, Sophie Barbe. 2022. *Computational Protein Design with Automated Reasoning and Deep Learning.* MASIM workshop, Paris, France.

Marianne Defresne, Thomas Schiex, Sophie Barbe. 2022. *Computational Protein Design with Automated Reasoning and Deep Learning.* BioSynSys 2022, Paris, France.

Sophie Barbe, **Marianne Defresne**, Younes Bouchiba. 2022. *AI-powered and structure-based Protein Design.* Lecture and practical: iBio summer school 2022, Banyuls-sur-Mer, France.

Funding and computational resources

The thesis was funded by the EUR BioEco (grant ANR-18-EURE-0021). It was also supported by the French ANR through grant ANR-19-PIA3-0004 (ANITI). This work was performed using HPC resources from CALMIP (Grant 2022-P21025), Jean-Zay GENCI-IDRIS (Grant 2022-AD011013779) and the Genotoul GIS bioinformatics platform. We thank all of them for their support, without which this thesis would not have been.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
CFN	Cost Function Network
CO	Combinatorial Optimization
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
CNN	Convolutional Neural Network
CPD	Computational Protein Design
DFL	Decision-Focused Learning
DL	Deep Learning
DPBB	Double-ψ β-Barrel
DR	Discrete Reasoning
GM	Graphical Model
GMEC	Global Minimum Energy Conformation
GNN	Graph Neural Network
KBP	Knowledge-Based Potential
LR	Learning Rate
MAP	Maximum A Posteriori
MLP	Multi-Layer Perceptron
MRF	Markov Random Field
MSA	Multiple Sequence Alignment
NLP	Natural Language Processing
PDB	Protein Data Bank
PLL	Pseudo-log likelihood
RNN	Recurrent Neural Network
WCSP	Weighted Constraint Satisfaction Problem

Chapter 1

Background

This chapter presents all the notions and concepts that will be used in this thesis. Since this work is multi-disciplinary, at the crossroad between two branches of AI and applied to protein design, some background is introduced to make this document accessible to scientist with various backgrounds.

First, Section 1.1.3 gives basic notions on proteins, from sequence to function via structure. Second, Section 1.2.5 introduces Deep Learning through a simple example, then describes the four main ingredients required, namely data, neural networks, loss functions and optimizers. Third, Section 1.3.3 presents Graphical Models and how they are used for automated reasoning, with an emphasis on the particular case of Cost Function Networks.

1.1 Proteins

Proteins are sometimes described as the "machinery of life". Indeed, these biological macro-molecules perform a wide variety of functions in all living organisms.

Some proteins, named enzymes, catalyze chemical reactions and are therefore essential to cell metabolism. Others have a structural role, such as collagen in tissues or keratin in nails; or a mechanical function such as allowing muscles to move. Hormone proteins like insulin have a regulatory function. Some proteins ensure transport of other molecules in the cell and outside, others are involved in cell signaling, storage or immune response.

The function fulfilled by a protein is tightly linked to its 3D structure, and results from billion of years of evolution. These notions are briefly explained in this section, which is restricted to proteins with an ordered 3D structure.

1.1.1 Protein Sequence

A protein is a macro-molecule composed of a succession of amino acid residues [Branden and Tooze, 2012]. Each of the 20 natural amino acids share a common structure, called the main chain: a central carbon atom C_α, to which are attached an amine group (NH2), a carboxyl group (COOH) and a hydrogen atom. What differentiate one amino acid from another is its side chain, which is also tied to the alpha carbon. Two amino acids are linked together via a peptide bond between the carbon atom of the carboxyl group of the first amino acid and the nitrogen atom of the amino group of the second one (see Figure 1.1). Once bonded, an amino acid is called a residue. The succession of the residue main chain atoms is called the *backbone* of the protein.

Each amino acid is identified by a one-letter or a three-letter code (see Table 1.1). They are usually divided into groups based on the physico-chemical properties of their side-chain. Among the various classifications that have been proposed [Taylor, 1986], there is a partition between hydrophobic side chains, charged residues and polar side chains. Finer classes can be proposed, including size-based (large or small), aliphatic or aromatic and positively or negatively charged. It is worth mentioning two special cases. The glycine has a single hydrogen atom as a side chain, and therefore specific properties. The proline, whose side chain connects to the backbone nitrogen, confers rigidity to the backbone.

The peptide bond is planar and rather rigid, but rotations of the backbone are possible around the $C - C_\alpha$ bond and the $C_\alpha - N$ bond (see Figure 1.1).

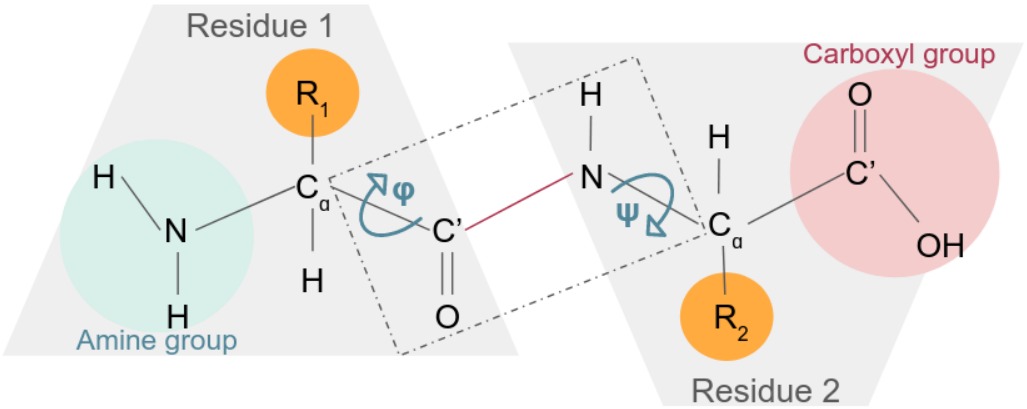


Figure 1.1: Each amino acid is composed of an amine group and a carboxyl group around a central carbon atom, as well as a variable part called the side chain (in orange). Two amino acids bond together via a peptide bond (in red) and they are then called residues. The peptide unit (in dotted line) is planar. The backbone is flexible via rotation around ϕ and ψ angles (in blue).

They are measured by dihedral angles denoted ϕ and ψ, respectively, and they are the main factor of *backbone flexibility*. A third dihedral angle, ω, measures the rotation around the peptide bond $N - C$ itself and is usually equal to π radians (trans conformation), although it can sometimes take the value 0 (cis conformation).

Side chains are also flexible: they can move in 3-dimensional (3D) space through rotations around the side-chain covalent bonds. The number of degrees of freedom varies with the type of amino acid. They are described by dihedral angles noted χ_i with $i \in \{1, 2, 3, 4\}$ depending on the type of amino acid. As a consequence, different conformations of the side chain are possible. The most common conformations are called rotamers, and they are described in dedicated libraries [Dunbrack Jr and Cohen, 1997].

1.1.2 Protein Structure

A Hierarchical Organization

The 3D structure of a protein can be described in a hierarchical way (see Figure 1.2). First, the amino acid sequence of a protein is also called its *primary structure*. The residues of the sequence arrange themselves into local *secondary structure* elements, joined by flexible loops. There are 2 dominant types of secondary structure elements, α-helices and β-sheets, whose shape is

Table 1.1: A possible classification of canonical amino acids, based on the chemical properties of their side chain.

Class	Name	3-letter code	1-letter code
Hydrophobic	Alanine	Ala	A
	Valine	Val	V
	Leucine	Leu	L
	Isoleucine	Ile	I
	Phenylalanine	Phe	F
	Methionine	Met	M
Charged	Aspartic acid	Asp	D
	Glutamic acid	Glu	E
	Lysine	Lys	K
	Arginine	Arg	R
Polar	Threonine	Thr	T
	Cysteine	Cys	C
	Asparagine	Asn	N
	Glutamine	Gln	Q
	Histidine	His	H
	Tyrosine	Tyr	Y
	Tryptophan	Trp	W
Special cases	Glycine	Gly	G
	Proline	Pro	P

due to regular patterns of hydrogen bonds between backbone atoms.

Secondary structure elements further organize themselves into a stable and functional 3D structure, called the *tertiary structure*. The overall topology — or fold — is determined by the way secondary structure elements connect together. A part on the polypeptide chain that adopts a stable and independent fold is called a domain. The structure is stabilized by interactions between residues brought into contact when the protein folds. Those interactions include hydrogen bonds, electrostatic and van der Waals interactions, and for some proteins disulfide bridge (a covalent bond between two cysteins). Hydrophobic residues unfavourably interact with water, the most common environment of protein. Therefore, hydrophobic interactions tend to create proteins with a core of densely packed hydrophobic residues, and a surface composed of hydrophilic (polar or charged) side chains.

Several polypeptidic chains can assemble to form a functional complex. In this case, their position is described by the *quaternary structure*. Such proteins are called multimeric, by opposition to monomers, composed of a single chain.

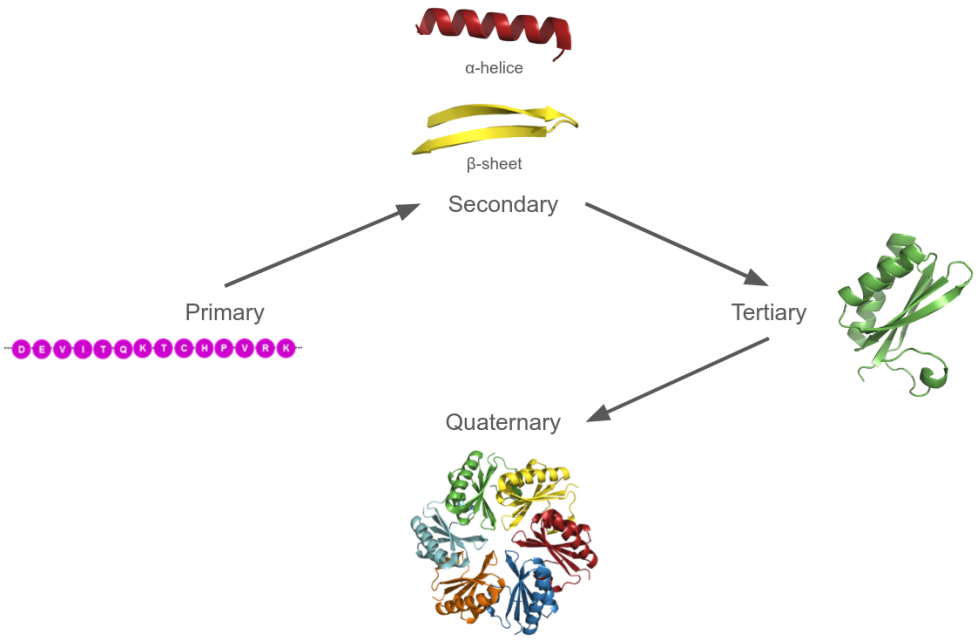


Figure 1.2: Protein structural organization is hierarchical. First, the unfolded protein sequence forms local secondary structure elements. Then those elements arrange themselves into a stable and functional 3D structure. Several chains can connect to form a complex called quaternary structure.

Proteins can be classified into 3 groups based on their tertiary or quaternary structure: globular, fibrous and membrane proteins. Globular proteins are soluble and they fold into an approximately spherical shape, hence their name. Fibrous proteins, such as collagen and keratin, have a structural role. Membrane proteins are part of cell membranes, and therefore they fold into a lipidic environment rather than water. Some of them are transmembrane: only a part of the protein lives in the membrane, while the rest has a soluble environment. Finally, some proteins are intrinsically disordered, meaning that they have an ordered structure only in presence of a binding partner [Dunker et al., 2001]. This work only covers ordered proteins, and it mainly focuses on globular proteins.

The 3D structure of a protein or a complex can be determined experimentally using X-ray crystallography, Nuclear Magnetic Resonance (NMR) or cryogenic electron microscopy (Cryo-EM) [Bhella, 2019]. They are usually

stored into the Protein Data Bank (PDB) [Berman et al., 2000], an open-access database. As of May 2023, the PDB contains $177,000$ structure, sometimes redundant, and for each of them, the Cartesian coordinates of each atom are available. Globular proteins being easier to crystallize than membrane proteins [Walian et al., 2004], they are over-represented in the PDB.

Folding and Flexibility

The sequence of a protein almost completely determines its native structure (for a given environment), as shown by Anfinsen's milestone experiment [Anfinsen, 1973]. He postulated his thermodynamical hypothesis: the native conformation is the most thermodynamically stable shape (in the intracellular environment). More precisely, the sequence folds into a conformation with minimal free-energy to ensure the overall stability of the protein.

However, the exact relationship between sequence and structure is yet to be established. Even if the best structure prediction algorithms have reached experimental accuracy [Jumper et al., 2021], they are not interpretable and therefore do not explain the folding mechanisms. Yet, Levinthal's paradox implies such mechanisms exist [Levinthal, 1969]. Indeed, an unfolded polypeptide chain has many degrees of freedom and therefore an astronomical number of possible conformations (approximated to 3^{198} for a small protein of size 100). If a protein had to randomly explore all possible conformations to fold, it would take longer than the age of universe. Yet, folding takes between 10^{-9} and 10^{-6} seconds, thus suggesting the existence of guiding mechanisms.

The 3D structure of a protein being stable does not imply it is rigid. Protein dynamics is due to local flexibility and collective movements. Local flexibility is due to the continuous motion of each atom which animates the protein with small-temperature-dependent breathing movements. Structure can also undergo significant collective motion, for instance due to electron transfer or large conformational changes between different final states of the protein, a phenomenon called allostery [Tsai et al., 2009]. Allostery is essential for many functions. A classic example is haemoglobin, that switches state upon binding to a dioxygen molecule to transport it in the blood.

1.1.3 Evolution and Function

Existing natural proteins result from billions of years of evolution. Random changes in DNA sequences create random mutations in protein sequences, which have been selected based on the functional advantage provided to the

organism producing the protein. The function of a protein is tightly linked to its 3D structure, which is determined by its sequence.

Homology

Homologous proteins, that have evolved from a common ancestor, often present similar function and structure. More precisely, the core region is usually conserved, while there is variability in the loop regions connecting secondary structure elements. Homology can be detected by a significant similarity in the amino acid sequence of two or more proteins. Two sequences are similar (respectively identical) at a given position if they have a similar (resp. identical) residue. Similarity between residues can be quantified by substitution matrices such as BLOSUM62 [Henikoff and Henikoff, 1992].

Algorithms, such as BLAST [Altschul et al., 1990], are used to align sequences and compute their similarity. The information is stored into a Multiple-Sequence Alignment (MSA), a matrix in which each row corresponds to a sequence and each column to a position in the sequence. Above 30% identity, two proteins are considered to be from the same family, and therefore to have a similar fold and biochemical function.

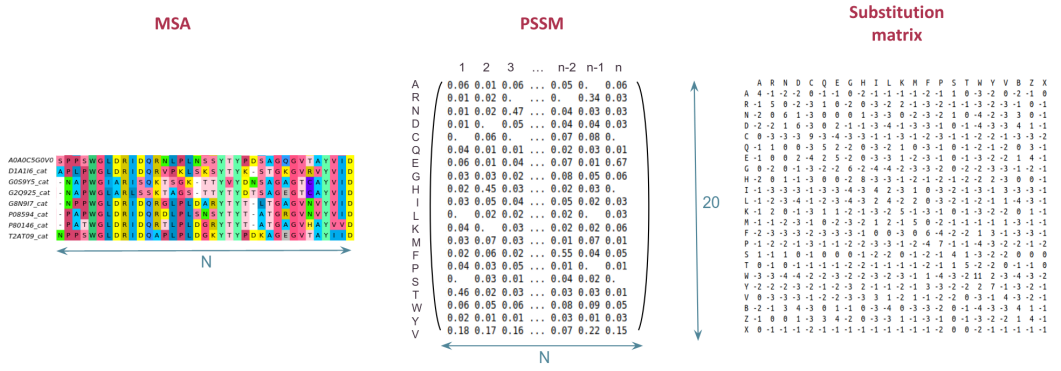


Figure 1.3: A Multiple Sequence Alignment (MSA) contains one sequence per row, and each column is a position in the amino acid sequence, with N the length of the sequence (left). A Position-Specific Scoring Matrix (PSSM) gives the probability of each type of amino acid at each position; therefore, it is a matrix of shape $20 * n$ (middle). A substitution matrix, here BLOSUM62 (right) is independent of the protein or the family considered. It indicates the similarity between residues.

Much information can be derived from the evolutionary information contained in MSAs. First, single-residue-level information can be obtained by building a position-specific scoring matrix (PSSM). Each column corresponds again to a position, and each row corresponds to one type of amino acid. The

PSSM gives the frequency of each residue at each position: a residue conserved at a given position suggests it is a key residue for the function of the protein or the stability of the structure.

Information on pairs of residues can also be derived from MSA. The detection of co-evolving residues, *i.e.*, that mutate in response to each other mutations, is a tool to detect contacts in the 3D space [Morcos et al., 2011; de Oliveira and Deane, 2017; AlQuraishi, 2019]. Indeed, co-evolutionary couplings suggest interaction and therefore proximity in space. Figure 1.3 illustrates a MSA, a PSSM and a substitution matrix.

Classification of Protein Architectures

Many amino acid sequences fold into a similar 3D structure. Indeed, for a small protein of 150 residues, there are around 10^{195} possible sequences (many of which would not fold), while the number of known folds for natural proteins is around $1,200$ [Chandonia et al., 2019].

A domain can be seen as either a structural, evolutive, folding or functional unit. Therefore, they are several ways of classifying them, based on both sequence, structure and sequence similarity. Two main classifications exist: SCOPe [Chandonia et al., 2019] and CATH [Sillitoe et al., 2015]. In the Structural CLassification of Proteins (SCOPe), domains are classified based on different levels of similarity: the class (for instance all α-helices or all β-strands), the fold (same topology), the superfamily (same fold but low sequence identity) and the family (same fold and high sequence identity). The acronym CATH stands for the four hierarchical levels used: the class, the architecture, the topology (same as fold in SCOPe) and homologous superfamily (similar function). Although very similar, these two classifications complement each other and allow to give a fairly comprehensive view of the classification and the links between the known domains [Hadley and Jones, 1999].

Protein Interactions

The main characteristic of proteins is their ability to bind to other molecules, which allows them to fulfill their biological function. They can bind to other proteins, to small molecules called a ligand or to DNA. Protein-protein interactions result in complexes (quaternary structure) that can contain thousands of amino acids. The contact area is called the interface and it is usually highly specific. The specificity of the interaction is determined by the tertiary structure of the proteins in contact and the position of the residue side chains at the interface. Binding to a ligand occurs at a small region of the protein surface

called the binding site, which is is also highly specific as it recognizes only one (or very few) type of molecule.

Among the many functions fulfilled by proteins, one of particular interest is catalysis, performed by enzymes. They catalyze chemical reactions in the cell and thus they are heavily involved in cell metabolism. An enzyme is highly specific as it usually only catalyzes one type of reaction at a given site. Catalysis happens upon binding to a substrate (one reagent of the reaction). The binding site and the catalytic site form the active site. Some enzymes requires another chemical species, called co-factor, to be active. As any catalyst, an enzyme accelerates the reaction but it is not modified by it. Therefore, it is reusable and thus of high interest for green chemical processes, by opposition to classical synthetic chemistry which uses more polluting catalysts.

1.2 Deep Learning

In this section, we describe the Deep Learning notions necessary to understand the various approaches proposed for protein design, and detailed in Section 2.3.3.

Deep Learning (DL) is a sub-category of Machine Learning aiming to extract patterns from data [Lecun et al., 2015]. DL is particularly efficient to process raw inputs into features, therefore by-passing the tedious and complex task of features hand-crafting. It has shown unprecedented performances on various kind of complex inputs, including images [Lecun et al., 1998], language [Vaswani et al., 2017; Kenton and Toutanova, 2019] and protein structures and sequences.

Schematically, the learning process relies on 4 main ingredients:

1. a large amount of data
2. an artificial neural network processing the data
3. a learning objective, called the loss function
4. an optimization process

Each of them will be described in a dedicated section. First, in order to illustrate the interplay between those 4 ingredients, we introduce the simple and historic example of the Multi-Layer Perceptron.

1.2.1 The Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP), also called fully-connected or feed-forward neural network, is the most basic of neural net architectures. Each layer takes as input a fixed-size vector $x \in \mathbb{R}^{d_0}$, then produces another vector $y \in \mathbb{R}^{d_1}$ such that

$$y = Ax + b$$

with $A \in \mathbb{R}^{d_0 \times d_1}$ and $b \in \mathbb{R}^{d_1}$. Then a non-linear function, called an *activation function*, is applied to y.

$$z = \sigma(y)$$

Each output of σ is called a neuron. The most usual activation functions are the sigmoid and variants of the Rectified Linear Unit (ReLU) (plotted in Figure 1.4).

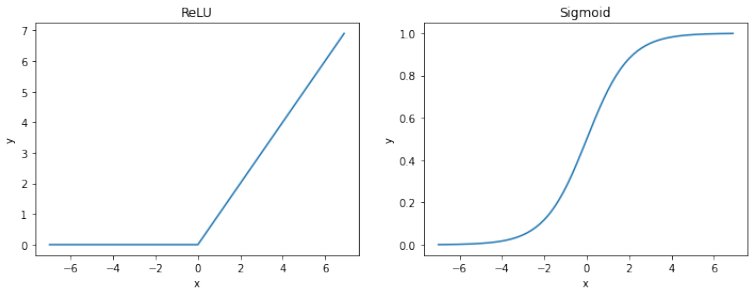


Figure 1.4: Plot of common activation functions: the ReLU (left) and the sigmoid (right). Their goal is to apply non-linearities between neural layers.

Several such layers are stacked together to form the complete MLP. The output of the final layer (either a real number or a vector of real numbers) is used to produce an answer to the targeted task. The learning process consists in adjusting the matrices parameters (called *weights*) so that the output reliably produces a correct answer. In the supervised setting, learning is achieved by processing a large collection of input vectors with associated known solution (or label) and minimizing an objective quantified in the *loss function* using some variant of gradient descent.

The global parameters of the MLP architecture, such as the number of layers (the depth of the neural net), the number of neurons (d_1, the width) or the activation function also impact the overall performances. They are called *hyperparameters* and they also need to be tuned to achieve the best performances.

Several criteria can be used to define the performance of a neural network. It can be a metric assessed on the validation (for hyperparameter tuning) or the test set, such as precision or mean error. The metric (or set of metrics) is chosen based on the task of interest. Other objectives can be targeted, including improving the speed of the prediction or the size of the model. Finally, performance can be assessed regarding data: a neural net is *data-efficient* if it requires few data for training, and it *generalizes* well if its test metric is also high on unseen data (see Subsection 1.2.5). All of these criteria to assess performance are task-dependant, and they are decided based on what the neural net will be used for.

Since training a neural net requires to process many times a large set of data, efficient computations are required. It is achieved via parallel computations: several inputs are processed simultaneously using a variable number of Graphics Processing Units (GPUs). Those inputs are stacked into tensors, *i.e,*

multi-dimensional matrices. Any data to be processed by a neural net must be expressed as a tensor.

MLPs are universal approximators [Hornik et al., 1989], meaning that they can approximate any continuous function as precisely as desired (given enough data and neurons). However, they do not take into account the specificities of the problem (for instance, invariance: a protein structure that is rotated has different coordinates but it stays the same structure), which limits their performances in practical settings where both the amount of data and the model size are limited. Therefore, various architectures have been developed and dedicated to a specific type of data, including images, graphs or sequences (see Section 1.2.3). Nevertheless, training any neural network always requires the elements described in this section, namely data, a specific architecture, a loss function and an optimization process. Each of them will be detailed in the following sections.

1.2.2 Data

A critical element for the overall performances of a neural network is the amount of available data and the way they are represented. Neural networks have been successful on many complex data types, including images, language and structured data such as graphs [Wu et al., 2021]. In all cases, both input and output data must be expressed as tensors. Mathematically, input data are described as a sample drawn from a probability distribution (almost always unknown).

These data are split into 3 subsets: training, validation and test sets. The training set is used to learn the weights of the neural net. The validation set is used to tune the hyperparameters and possibly to decide when to stop training. Once all parameters are chosen, the performances of the neural net are assessed on the test set, chosen to be representative of the target task. If the goal is to generalize to unseen data, the three datasets should be as independent as possible.

In the supervised setting, the input data is associated with a ground truth label to be recovered. For instance, images can be associated with an object present in the image [Deng et al., 2009]. In this case, the neural network learns by imitation. In practice, labelled data are less common than unlabelled data, handled by unsupervised methods. The last major training setting is Reinforcement Learning [Kaelbling et al., 1996], where an agent learns by interacting with its environment. It will not be discussed further in this work.

The specificity of the output also impacts the setting. The output of a predictive model can be interpreted as an information over the input. If it is a real number, the model performs a regression: if it is a class, the model performs a classification. For instance, for an input protein sequence, its stability could be predicted (regression), or whether it is intrinsically disordered or not (classification). The output of a generative model is a new sample, drawn from the (unknown) input distribution. For instance, from protein sequences of a given family, one may want to generate a new sequence of a protein that would belong to this family. A summary of the different settings based on the kind of input and output is proposed in Figure 1.5.

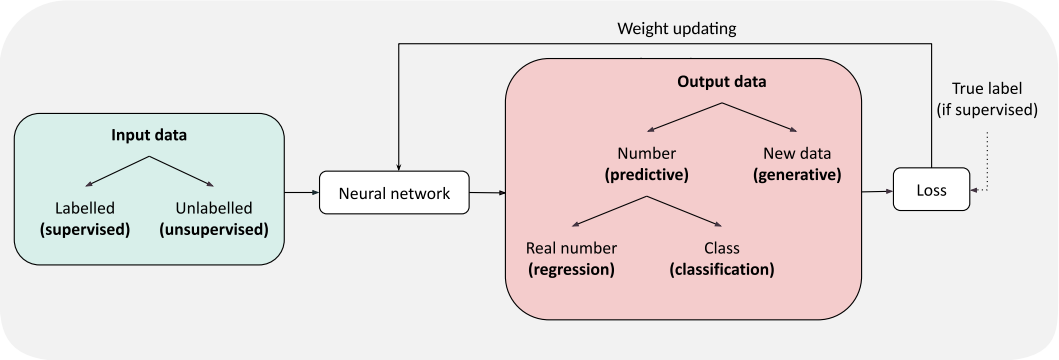


Figure 1.5: Existing settings (in bold) based on the kind of input (in green) and output (in red) data.

1.2.3 Architectures

The type of data to process, as well as the desired output (prediction or generation of a new sample), drive the choice of the best-suited architecture of neural network. Neural nets are especially useful to process raw data; when highly-informative features are available, classic ML methods such as logistic regression or SVM are often sufficient. Therefore, this section is organized around the type of data to process: first, images processed by Convolutional Neural Network (CNN), then sequences by recurrent architecture and attention models and graphs by Graphs Neural Nets (GNN). Finally, some architectures for generative models are described.

Convolutional Neural Network

Basic MLPs have been refined to process images by restricting their linear transformations to local convolutions [Lecun et al., 1998]. This operation com-

putes a pixel state as a linear combination of neighbouring pixels only. Through convolution, the output of a translated image is just translated. Convolutions are interleaved with *pooling* layers that merge blocks of (usually 2×2) pixels, thus reducing scale. The succession of convolutive and pooling layers, separated by activation functions, can extract more and more global features while taking into account neighbouring information. Figure 1.7 summarizes the resulting Convolutional Neural Network (CNN).

CNNs are an example of an architecture leveraging the symmetries of the problem (*i.e*, a translated motif stays the same motif) through *translation-equivariance* (translation of the input results in a similarly translated output) [Matsugu et al., 2003]. Invariance can exist for all sorts of operations on the input, including rotations and permutations. An architecture that transforms its output in the same way as the input is said equivariant. Both invariance and equivariance are a major target in Deep Learning since they reduce the number of parameters to be learned without losing any power of representation. Less data is needed and performances improve in terms of training time, data-efficiency and generalization ability.

Empirically, CNN accuracy was observed to increase with the number of layers used (the depth), but this eventually lead to numerical issues during back-propagation, which slows down and possibly stops learning. To go deeper, *residual connections* [He et al., 2016] were introduced: the input of a layer is directly added to the output of following layers (see Figure 1.6).

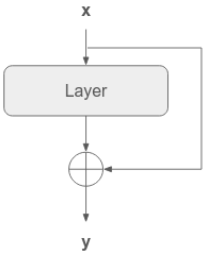


Figure 1.6: Residual connection.

Finally, while CNNs exploit translation-equivariance, they are sensitive to rotation of the input. Thus, a rotated motif in the input - possibly a protein structure - will not be automatically recognized. A usual approach to tackle this issue is to use *data augmentation* [Shorten and Khoshgoftaar, 2019]: the training set is completed by similarly perturbed (rotated) images. However, this approach makes training longer and harder and using rotation-equivariant architectures is better.

Recurrent Architecture and Attention Models

Text is a very common type of data, that has been massively analyzed in Natural Language Processing (NLP), driving the development of dedicated architectures. From existing unlabelled sentences, widely available, training

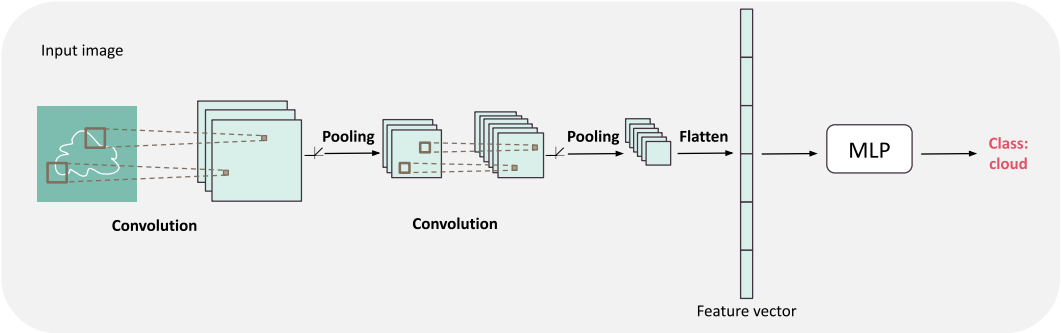


Figure 1.7: Convolutional neural network (CNN) architecture. The architecture is composed of a succession of convolutions, followed by an activation function and pooling layers to extract more and more global features from the image. Then,if the task is a classification or a regression, these features are flattened into a vector fed into a usual MLP which predicts the class or regression value.

can be supervised by next-word prediction or masking: completing a partial or masked sentence with an appropriate word. Protein sequences can be seen as sentences made of amino acid types, and thus can be processed by the same architectures.

Compared to images, that can always be scaled to a fixed format, sentences have variable length. Recurrent Neural Networks (RNN) have been developed to process each word at a time by the same operation. The current output is computed from the current input word and the previous output, hence the name recurrent (see Figure 1.8). The idea, which is reminiscent of Hidden Markov Models, is to integrate the information from previous words to predict the next one. To preserve information from previous words, additional parameters, acting as a form of memory, have been introduced in LSTMs (long short-term memory) [Hochreiter and Schmidhuber, 1997].

Individual words must be embedded into fixed-size numerical tensors (vectors here). The most obvious approach is to sort all words and identify each by its position i. But close numbers may correspond to semantically unrelated words and this makes training difficult. One Hot encoding represents word by a vector of 0s, except for a 1 at the ith position.

A more informative representation uses an *embedding learned* by a Language Model (LM) [Mikolov et al., 2013; Le and Mikolov, 2014] to give them a semantic flavor. Similar embeddings (in terms of cosine distance between vectors) should have a similar meaning. Ultimately, they can even be used to get a working semantic "algebra" as often exemplified by the (approximately

satisfied) equation (king - man + woman = queen) [Vylomova et al., 2016]. Suitable learned embeddings are critical for the performances of various models built on top of them.

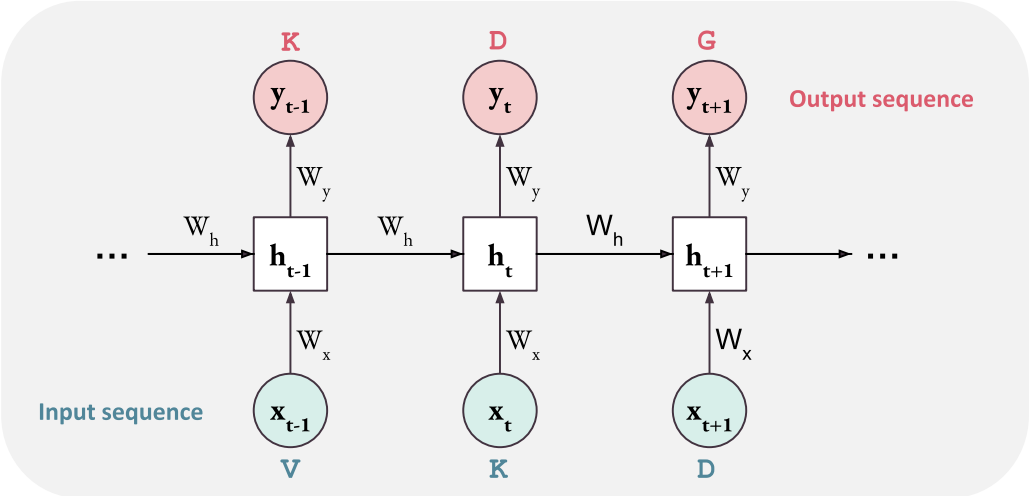


Figure 1.8: Pipeline of a Recurrent Neural Network. Elements x of the input sequence are processed individually by the same weights W.

A more recent approach to handle sequential data replaces recurrence by so-called *attention mechanisms*, popularized in the Transformer architecture [Vaswani et al., 2017]. Recurrent architectures have an inherent difficulty in exploiting long-distance interactions in a sequence [Hochreiter et al., 2001]. Transformers have been precisely designed to fight this limitation by leveraging self-attention to identify which part of the input is important (and how) for the prediction. The neural net can then focus on those parts, no matter their range, to predict the output. For instance, when predicting protein contacts, attention can make each residue attend to only few other residues, which are therefore more likely to truly be in contact [Bhattacharya et al., 2020].

To allow for considering any distant interaction, the size of the input sequence is bounded to a maximum and data processing is parallel instead of being sequential. This makes the process potentially more efficient but also more memory intensive, especially for large maximum lengths.

Graph Neural Networks

Previous architectures focused on numerical data, which can be directly represented as tensors. However, it excludes many structured data, such as interaction networks, molecular structures or meshed surfaces [Bronstein et al., 2017].

Therefore, a dedicated architecture is required; this section will focus only on graphs, but Geometric Deep Learning has extended the developed models to other structured non-Euclidean data [Bronstein et al., 2021].

One of the first application of Graph Neural Networks (GNN) was on molecules [Gilmer et al., 2017], that can naturally be represented as graphs: each atom corresponds to a vertex and the covalent bonds are edges. Each layer of the proposed GNN consists in 2 operations, as summarized in Figure 1.9: *message passing* and *graph pooling*.

Message passing, first introduced for graphical models [Pearl, 1982], is formally described in equation 1.1. Each node v is associated with a state h_v, initialized from its input features or randomly. During one iteration of message passing, each node updates its state based on its previous state and messages received from their neighbouring (*i.e.,* connected) nodes. These message are aggregated based on a parametric (*i.e.*, learnable) function f, that can also depends on node and edge features [Wu et al., 2021].

$$h_v^{(t)} = \phi(h_v^{(t-1)}, \sum_{u \in N(v)} f(h_u^{(t-1)})) \tag{1.1}$$

Following message passing, a graph pooling operation reduces the size of the graph by fusing nodes together. The mathematical details of the message-passing and graph-pooling operations result in variations of GNN architectures. Notable ones include attention [Velickovic et al., 2017] or relational reasoning [Palm et al., 2018].

GNNs suffer from some limitations. First, message passing requires many-operations to connect distant nodes, which limits their ability to model long-range dependencies. Second, their expressive power is limited as they are unable to distinguish isomorphic graphs in some cases [Morris et al., 2019].

Very few alternatives to process graphs without message passing have been proposed [Liu et al., 2021; Hu et al., 2021]. We based our architecture on the work by Liu et al., a MLP augmented with a gating mechanism acting like attention to focus on the most relevant part of the input [Liu et al., 2021; Hu et al., 2021].

Architectures for Generative Models

Generative models aim at learning the unknown distribution of the training data in order to generate new data from the same distribution. A simple gener-

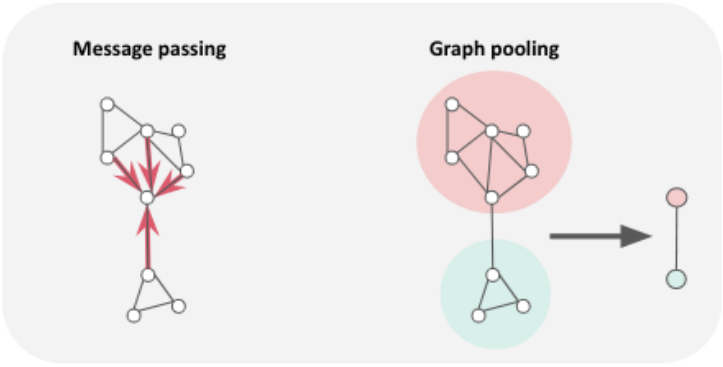


Figure 1.9: One layer of a Graph Neural Network. First, each node sends a message to its neighbours and updates its own state based on the messages received. Then, the graph is reduced using a specific pooling operation.

ative model can be learned using auto-encoders: an input x is successively reduced in dimensionality by an encoder, producing an internal low dimensional representation of the input x (a latent representation). This representation is then decoded to produce back the original input x. This encoder-decoder pair is trained as a single network that must learn the identity function. This process has been simplified in *Variational Auto-Encoders* (VAE) [Kingma and Welling, 2014] with a simpler latent distribution representation. The learned latent representation may be useful as a learned embedding of the input but also for generative purposes: new data can be generated by sampling the latent space and decoding it. However, this often leads to inconsistent output because the fraction of the latent space that describes correct output may be very tiny.

Another popular approach is based on *Generative Adversarial Networks* (GAN) [Goodfellow et al., 2014]. GANs also use a combination of two neural nets: a generator that learns how to generate new data (from the same distribution as the training data) and a discriminator that learns to predict whether its input is out-of-distribution or not. The training objective, encapsulated inside the loss, encourages the generator to fool the discriminator and the discriminator to reject out-of-distribution data, hence the name adversarial. After training, the generator alone is used to generate new data. Training GANs can be challenging as both learners need to learn at comparable speed [Arjovsky et al., 2017]. GAN and VAE are unsupervised methods but the inner architecture of their networks are of the same type as discussed before (MLP, CNN, attention-based), depending on the type of data to handle.

More recently, diffusion models have been proposed, initially to generate images. They work in two steps. First, during the forward diffusion stage, the data is perturbed by adding more and more Gaussian noise. In the reverse stage, the model is trained to reverse the diffusion process and predict back the input data [Croitoru et al., 2023]. Once trained, those models can generate new samples by denoising a random input, and they usually produce high-quality and diverse samples. They have now been applied to other kind of data, including protein structures [Watson et al., 2022; Anand and Achim, 2022].

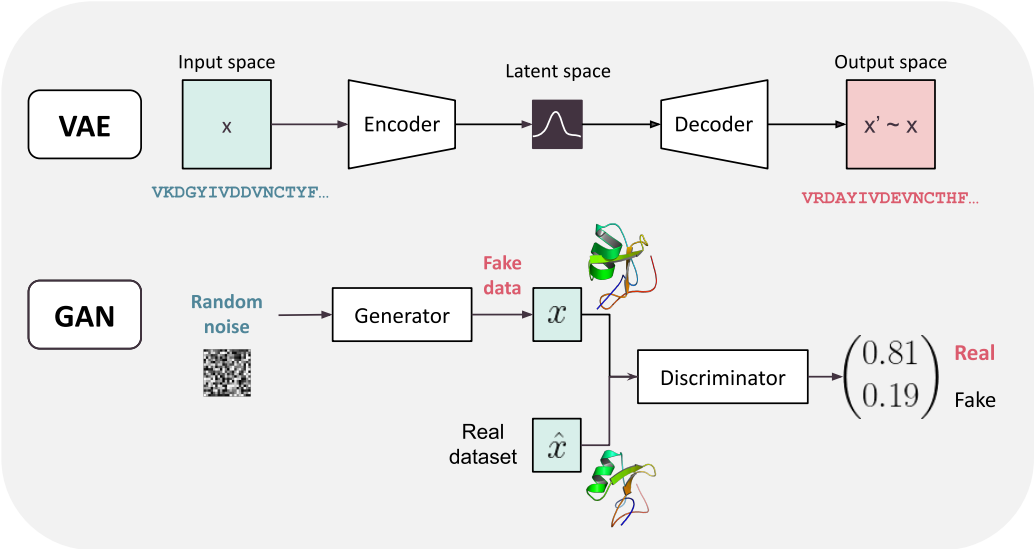


Figure 1.10: Examples of generative models. From top to bottom: Variational Auto-encoder (VAE) and Generative Adversarial Network (GAN) models. The encoder, decoder, generator and discriminator may be neural networks of any type (including MLPs, recurrent and attention-based).

1.2.4 Loss Function and Optimization

Loss Function

A numerical loss function quantifies how well the learning objective is reached, based on known answers (supervised case) or not. At each training iteration, a sample is processed by the network into a predicted output. The loss on this output is computed, then its gradients with respect to each of the net's weight are computed using backpropagation, a technique based on the chain rule [Lecun et al., 2015]. Finally, the weights are updated using gradient-descent-like method.

The choice of the loss function is critical to the overall performance. Since neural training is based on its gradients, the loss function must be differentiable. It is obviously chosen based on the problem setting. Regression tasks often use mean absolute or squared error. For classification, the output vector is turned into a probability over each class, then its cross-entropy is used as loss. Unsupervised methods require specific loss function, such as the likelihood [LeCun et al., 2006]. Beyond these very common tasks, the loss is chosen on a case-by-case basis.

Beyond supervised or unsupervised models, intermediate settings exist, such as self-supervised learning [Doersch and Zisserman, 2017], where a supervision signal is built out of unlabelled data, and *transfer learning*. During transfer [Pan and Yang, 2010], an already-trained model is reused on a new task. It is very common for Language Models: a model is first trained on very large corpus of texts to produce good-quality learned embedding. Then, those embeddings are used as inputs of a downstream task. An existing network can also be *fine-tuned* on a smaller, more-specific dataset. It is used as good initial weight to train on the task of interest (with a smaller learning rate, or with some layer frozen). Finally, it is possible to train on several objective at the same time, a method called multi-task learning. It usually improve performances on each individual task as well as generalizability [Caruana, 1998; Baxter, 2000] (see Subsection 1.2.5).

Optimization by Gradient Descent

Learning is the process of updating the neural net weights such that the loss is minimized. Formally, in the supervised setting, input X and target Y are drawn from a probability distribution P. The goal is to learn a function matching X to Y. This function f_θ is parameterized by the weights $\theta \in \mathbb{R}^p$ of the neural net. The statistical problem is

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{X,Y \sim P} L(Y, f_\theta(X))$$

In practice, it is solved by Empirical Risk Minimization: the empirical expectation of the loss over the training set is minimized.

$$\min_{\theta \in \mathbb{R}^p} \sum_{x,y \in \text{trainset}} \frac{1}{|\text{trainset}|} L(y, f_\theta(x))$$

The optimization is based on gradient descent. In its basic form, the gradients of the loss L is computed with respect to each weight w, then the weights

are updated as follows:

$$w^{t+1} = w^t - \eta \nabla_w L(w)$$

The parameter η is called the learning rate (LR) and it is critical for the training convergence. If too small, convergence will be slow; if too big, even a local minimum will not be reached.

The default gradient descent is batch gradient descent, where the gradients over all training examples are computed and averaged to take one step of descent. It converges directly to the best expectation of the gradient that can be computed, but it is costly for large dataset. Many variants have been proposed to improve convergence speed and performances [Ruder, 2016]. Notable improvements include mini-batch gradient descent, where a fixed number of training examples are used for one update. Another variant is the stochastic gradient descent (SGD), where only one example is processed to take a descent step. It achieves faster iteration, but has a lower convergence rate [Bottou and Bousquet, 2007].

Another very popular optimizer is Adam [Kingma and Ba, 2015], whose name is derived from adaptative moment estimation. While SGD uses a single learning rate for all weights during all training, Adam computes adaptative LR for different weight based on estimations of first and second moments of gradients. The momentum avoids being stuck into local minima.

A major issue during training is *exploding or vanishing gradient*. Indeed, too big or too small gradients, often resulting from very deep architecture, slow or even stop convergence. It can be partly dealt with with input normalization: avoiding to have big and small weights (thus small/large updates) may prevent from staying stuck into a plateau region. A step further is batch normalization, that normalizes the input of each hidden layer. Another tool against vanishing gradient is a good weight initialization. Commonly used are Xavier's initialization [Glorot and Bengio, 2010] on layers with symmetric activations, and Kaiming's initialization [He et al., 2015] for ReLU.

1.2.5 Generalization Ability

The goal when training a neural network is to perform well (in terms of metric, time, data-efficiency, ...) on the task of interest. This task is simulated by the test set, which should be as representative as possible (*i.e.,* drawn from the distribution of interest). For instance, a DL model trained to classify medical images from a specific device may perform poorly on images from other devices [Liu et al., 2019]. In this example, the test set should have contained images from all the devices the model was aimed to process.

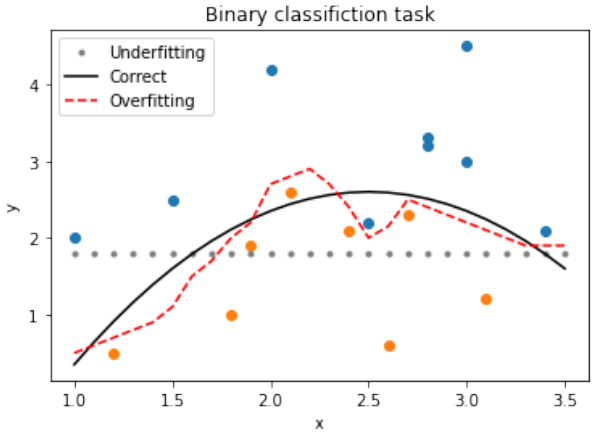


Figure 1.11: Illustration of over-fitting on a binary classification task.

The DL model may be aimed to perform well on any data, including unseen one. This is known as *generalizability*. It is measured on the test set, which (in this case) should be as different as possible from both training and validation set. Otherwise, reproducibility issues may arise [Kapoor and Narayanan, 2022]. In the previous example, performances were good only on images from one device, but they could not be reproduced on other devices. This illustrates the importance of data being drawn from the distribution of interest (in this case, images from many devices and not a single one).

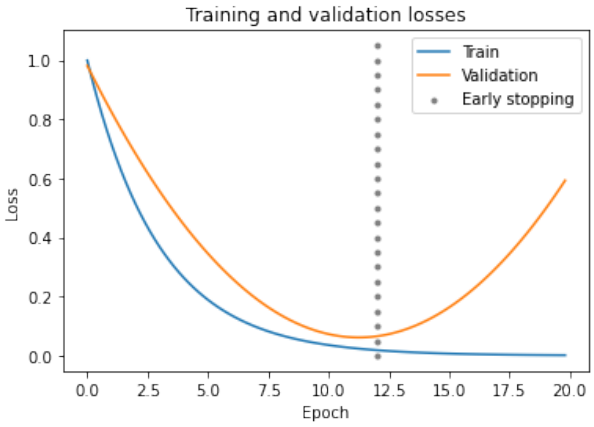


Figure 1.12: Illustration of early stopping to avoid over-fitting. When the loss assessed on an independent validation set starts increasing, training should be stopped.

Generalizability issues can also result from over-fitting. It occurs when the model fits the training data so tightly that it learns the dataset and therefore

the model performs poorly on independent data (see Figure 1.11). A few guidelines can help reduce overfitting. In accordance with the *parsimony* principle, the chosen neural architecture should be the simplest that fits for the best results. Some mechanisms can be added, such as the popular dropout [Srivastava et al., 2014] that randomly deactivates some neurons during training. To simplify the model and avoid over-fitting, *regularization* on the neural net's weights can be added in order to encourage weights to be small. It is automatically handled by the optimizer as the weight decay. Finally, over-fitting can be limited by stopping training early: when the validation loss starts increasing, it indicates the generalization error is increasing and therefore training should be stopped. It is illustrated in Figure 1.12.

1.3 Graphical Models for Automated Reasoning

The previous section described deep learning, one branch of Artificial Intelligence (AI). This section deals with an another aspect, *reasoning*. It focuses on Graphical Models, a family of models that encompass many NP-hard reasoning and optimization frameworks, both deterministic and stochastic.

After introducing graphical models, this section focuses on a specific framework to model numerical functions, Cost Function Networks. Definitions and examples are given, then the solvers we used are introduced.

1.3.1 Discrete Graphical Models

Graphical Models (GMs) are a family of models aiming to describe functions of many variables using decomposability [Cooper et al., 2020]. In this work, only discrete GMs (*i.e.,* describing a function over discrete variables) are considered, even though GMs with continuous variables, such as Gaussian GM [Uhler, 2017], exist. A joint function on all the variables of the set is defined by combining simpler functions, *e.g.* involving a small subset of variables. The combination is done by an associative and commutative operator, such as the addition, multiplication or the logical operations.

As suggested by their name, GMs can be represented graphically (see an example on Figure 1.13). Each variable is associated with a vertex, and edges are drawn between each pair of variables participating together in a function. The graphical representation makes it possible to describe a function on many variables concisely as one edge can be interpreted as one pairwise function.

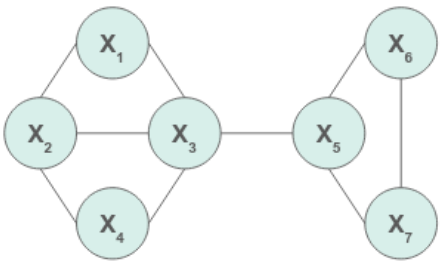


Figure 1.13: Graphical representation of a GM. Each variable X_i corresponds to a vertex and there is one edge if two variables participate together in a function.

Many reasoning and optimization frameworks can be described as discrete graphical models. GM with Boolean variables corresponds to propositional

logic, with one usual query being satisfiability problem (SAT). GM on finite-domain variable can describe functions that are Boolean (*e.g.*, Constraint Networks, with one query being Constraint Satisfaction Problem (CSP)) or finite-domain functions (detailed in the next subsection). All of these examples are deterministic. GMs can also describe discrete probability distributions using small real-valued functions. This includes the widely-used Markov Random Fields (MRF) [Koller and Friedman, 2009] and Bayesian Networks [Bishop and Nasrabadi, 2006]. Stochastic GMs are of particular interest because they can be learned from data.

Since GMs gather many settings, they have a wide variety of applications. Deterministic GM are used for knowledge representation, and learning and reasoning tasks. It includes scheduling [Baptiste et al., 2001] and planning [Van Beek and Chen, 1999] among many others [Dechter, 2013]. Stochastic variants are used in statistics, including physics statistics, and Machine Learning, with MRF being often applied to image processing such as segmentation [Li, 2009].

1.3.2 Cost Function Networks

Definition

Cost Function Networks (CFN) are the main type of GM that will be used in this thesis. They were chosen for their ability to represent both numerical and logical functions.

For clarity, in the rest of the document, sets, vectors and tensors will be usually denoted in bold. Formally, a CFN $\mathcal{M}$ is a triplet $(\mathbf{X}, \mathbf{D}, \mathbf{C})$ with:

- $\mathbf{X} = \{X_1, ..., X_n\}$ a set of **variables**
- $\mathbf{D} = \{D^1, ..., D^n\}$ a set of finite **domains**. For a subset of variables $\mathbf{S} \in \mathbf{X}$, the Cartesian product of all D^i with $X_i \in \mathbf{S}$ is noted $D^{\mathbf{S}}$.
- $\mathbf{C}$ a set of **cost functions** $c_{\mathbf{S}}$. Each depends on a subset of variables $\mathbf{S} \in X$ and $c_{\mathbf{S}} : D^{\mathbf{S}} \to \{0, ..., k\}$

Each cost function $c_{\mathbf{S}}$ is a numerical function bounded by an integer k (or possibly $+\infty$). Using as addition $a +^k b = \min(a + b, k)$, a joint cost function is defined by:

$$C_{\mathcal{M}}(\mathbf{x}) = \sum_{c_{\mathbf{S}} \in \mathbf{C}} c_{\mathbf{S}}(\mathbf{x}[\mathbf{S}])$$

where $\mathbf{x}[\mathbf{S}]$ denotes the projection of $\mathbf{x}$ on $\mathbf{S}$.

Once the joint cost function $C_{\mathcal{M}}$ is described by a CFN, many queries can be asked on $C_{\mathcal{M}}$. One of the most important is minimization, which is known as the Weighted Constraint Satisfaction Problem (WCSP). Formally, it aims to find an assignment $\mathbf{x}$ of all variables such that:

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{y} \in D^{\mathbf{X}}} C_{\mathcal{M}}(\mathbf{y})$$

The WCSP decision problem is NP-complete. It means computing the cost of an assignment is polynomial, but there is no known method to compute an optimal solution efficiently (*i.e.,* in polynomial time).

An important property of CFNs is them being invariant under scaling and shifting (*i.e.,* an affine transformation) of all the weights. Such a transformation does not change the optimal solution, but it changes of course the cost of the optimum. Thanks to this property, using discrete positive-valued cost functions is not restrictive in practice. Indeed, fixed decimal point numbers are handled by scaling, and negative number by shifting. In particular, maximization problems can be solved as well as minimization.

Constraints and Redundancy

A *constraint* is a cost function F such that $F(\mathbf{t}) \in \{0, \infty\}$: it exactly forbids all assignments $\mathbf{t}$ such that $F(\mathbf{t}) = \infty$. Said otherwise, a constraint either allows or forbid an assignment.

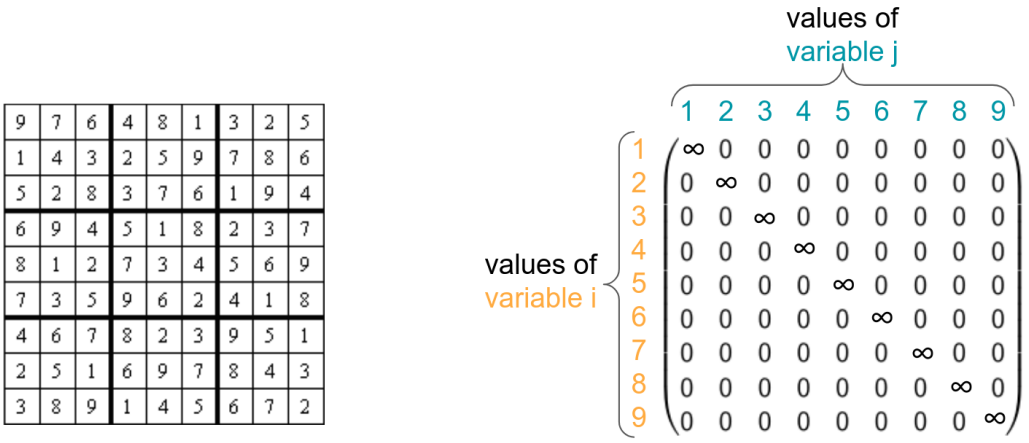
When a given function F is never larger than another function F' (denoted $(F \leq F')$), F is known as a *relaxation* of F'. If F and F' are constraints and such that $F \leq F'$, we say that F is a logical consequence of F'. Whenever F' is satisfied (*i.e.,* equal to 0), F is satisfied too.

For a set of constraints $\mathbf{C}$, $F \in \mathbf{C}$ is redundant w.r.t. $\mathbf{C}$ iff $\mathbf{C}$ and $\mathbf{C} \setminus \{F\}$ define the same function. At a finer grain, we say F is partially redundant if $\exists F' < F$ such that $(\mathbf{C} \setminus \{F\}) \cup \{F'\}$ and $\mathbf{C}$ define the same function.

Consider for example $\mathbf{Y} = \{Y_1, Y_2, Y_3, Y_4\}$ with domains $\{0, 1\}$ and $\mathbf{C} = \{Y_1 \neq Y_2, Y_2 + Y_3 > 1, Y_3 \neq Y_4\}$. No constraint is redundant in $\mathbf{C}$, but if Y_2 and Y_3 have already been assigned to 1, then the constraint $Y_2 + Y_3 > 1$ becomes redundant w.r.t. $\mathbf{C}' = \mathbf{C} \cup \{Y_2 = 1, Y_3 = 1\}$. In the context of $\{Y_1 = 0\}$, $Y_2 + Y_3 > 1$ becomes partially redundant, as it could equivalently be replaced by the weaker $Y_2 = Y_3$.

Example

We illustrate how to model problems into CFN on the example of Sudoku. Sudoku is a game consisting in filling a 9×9 grid with integers between 1 and 9 such that no identical figures are in the same row, column or square (see Figure 1.14 (left)). This problem only contains constraints, thus representable by Boolean cost functions. Such logical information is represented by a CFN with costs in $\{0, \infty\}$. Given a pair of variables, an infinite cost on a pair of values forbids the joint assignment of these values to the variables.



9	7	6	4	8	1	3	2	5
1	4	3	2	5	9	7	8	6
5	2	8	3	7	6	1	9	4
6	9	4	5	1	8	2	3	7
8	1	2	7	3	4	5	6	9
7	3	5	9	6	2	4	1	8
4	6	7	8	2	3	9	5	1
2	5	1	6	9	7	8	4	3
3	8	9	1	4	5	6	7	2

Figure 1.14: Left: a filled grid of Sudoku. No cells on the same row, column or $3*3$ square contains the same figure. These rules are constraints and they are encoded in a cost function (Right).

The CFN has 81 variables, each corresponding to one cell of the grid and with domain $\{1, ..., 9\}$. To prevent constrained variables (*i.e.,* corresponding to cells on the same row, column or square) from simultaneously taking the same value, a cost of ∞ will be given on assignments of an identical pair, and costs of 0 otherwise. Therefore, there will be a pairwise cost function on each constrained pair of variables. A convenient way to visualize pairwise cost functions is via tensors, as shown in Figure 1.14 (right).

The rules of Sudoku are redundant. Partial redundancy is obvious: when only one cell is empty, it can be guessed from the information of the row alone, without looking at the column or the sub-square. Some rules are also completely redundant, and removing them does not change the solution of the Sudoku grid. The minimal set of rules from Sudoku is still an open question, even if it has been proven that (at least) 162 over the 810 are redundant [Demoen and de la Banda, 2014].

Tools to Solve a CFN

In this work, we will use existing solvers mostly as black boxes to solve our WCSPs of interest. For exact resolution, we use the open-source solver toulbar2 [Allouche et al., 2015; Hurley et al., 2016], available at https://github.com/toulbar2/toulbar2.

In addition to providing the optimal solution(s), toulbar2 builds a proof of optimality. Nevertheless, solving becomes more and more expensive as the size of the problem increases: in the worst-case scenario, the complexity increases exponentially with the number of variables. However, the number of calculations can be arbitrarily limited via a parameter called backtrack [Harvey and Ginsberg, 1995]: solutions are provided in limited time, but their quality is not guaranteed.

Toulbar2 performances have been assessed independently on various competitions. It won several medals in competitions on Max-CSP (CPAI08, 2022 XCSP3) and probabilistic graphical models (UAI 2008, 2010, 2014, 2022 on the MAP task). For protein design (in the discrete pairwise formulation, detailed in Section 2.2.1), it was said to have "significantly improved the state-of-the-art efficiency" [Hallen and Donald, 2019]. Some of toulbar2's additional functionalities have been introduced specifically for designing proteins, including the production of guaranteed diverse high-quality solutions [Ruffini et al., 2019].

Still, protein design problems may have hundreds to thousands of variables, and therefore exact solving may become practically intractable. In those cases, an approximate probabilistic solver that scales better is attractive. We will use the Low-Rank Bloc Coordinate Descent (LR-BCD) method [Durante et al., 2022], which is based on a convex relaxation of the WCSP.

1.3.3 Probabilistic Interpretation of CFNs

CFNs as Markov Random Fields

CFN can be interpreted as a Markov Random Field, a type of stochastic GM [Koller and Friedman, 2009] for which the assignment of a variable corresponds to the realisation of a random variable. A discrete MRF $\mathcal{M}$ is defined on a set of discrete domain variables $\mathbf{X}$ and a set $\mathbf{C}$ of non-negative real-valued cost functions. It induces a joint probability distribution $P_{\mathcal{M}}$ defined by:

$$C_{\mathcal{M}}(\mathbf{x}) = \prod_{c_{\mathbf{S}} \in \mathbf{C}} c_{\mathbf{S}}(\mathbf{x}[\mathbf{S}])$$
$$P_{\mathcal{M}}(\mathbf{X} = \mathbf{x}) \propto C_{\mathcal{M}}(\mathbf{x})$$

One usual query on MRF is Maximum a Posteriori (MAP), *i.e.,* maximizing $P_{\mathcal{M}}$, which is equivalent to maximizing $C_{\mathcal{M}}$. If we take the negative logarithm of the probability distribution, we find back the joint cost function defined by a CFN:

$$\begin{aligned}\max_{\mathbf{x} \in D^{\mathbf{X}}} \prod_{c_{\mathbf{S}} \in \mathbf{C}} c_{\mathbf{S}}(\mathbf{x}[\mathbf{S}]) &\Leftrightarrow \max_{\mathbf{x} \in D^{\mathbf{X}}} \log(\prod_{c_{\mathbf{S}} \in \mathbf{C}} c_{\mathbf{S}}(\mathbf{x}[\mathbf{S}])) \\ &\Leftrightarrow \min_{\mathbf{x} \in D_{\mathbf{X}}} \sum_{c_{\mathbf{S}} \in \mathbf{C}} -\log c_{\mathbf{S}}(\mathbf{x}[\mathbf{S}])\end{aligned}$$

Therefore, solving a WCSP over a CFN (with infinite bound: $k = \infty$) is equivalent to solving MAP over a MRF. Since MRFs can be estimated from data [Taskar et al., 2004] (*i.e.,* finding the probability distribution of variable assignment), the probabilistic interpretation of CFN makes it possible to learn it from data as well [Brouard et al., 2020].

This probabilistic interpretation can be considered from a statistical physics point of view. In a MRF, the probability of a set of variables $\mathbf{X}$ to be assigned to a set of values $\mathbf{x}$ is considered. This probability can be written using Boltzmann's equation, which gives the probability of a system $\mathbf{X}$ to be in a state $\mathbf{x}$:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z(T)} e^{-U(\mathbf{x})/T}$$

where T is the system temperature and Z is the normalization constant (also called partition function). $U(\mathbf{x})$ is the potential energy of the system in state $\mathbf{x}$. Estimating the MRF is the same as finding U from data.

We saw that a MRF is equivalent to a CFN through a $-\log$ transform. Therefore, the energy $U(x_i)$ can be interpreted as a cost in a CFN. Thus, an infinite cost or energy corresponds to a zero probability, which is a forbidden assignment.

Estimating an MRF

One major interest of MRFs is that they can be estimated from data.

We consider a dataset $\mathbf{S}$ containing m values $(\mathbf{y}^l)_{1\leq l\leq m}$ of a sequence of variables Y. We assume these values to be i.i.d. samples drawn from an unknown probability distribution $P(\mathbf{Y})$. We want to find an estimate $P^{\mathcal{M}}$ of this distribution using a Markov Random Field $\mathcal{M}$.

A natural criteria for the GM $\mathcal{M}$ is the negative logarithm of the probability of the observed samples, or negative log-likelihood (NLL) :

$$\mathrm{NLL}(S) = -\log(\prod_{\mathbf{y}\in S} P^{\mathcal{M}}(\mathbf{Y}=\mathbf{y})) = -\sum_{\mathbf{y}\in S}\log(P^{\mathcal{M}}(\mathbf{Y}=\mathbf{y})))$$

However, this negative log-likelihood is intractable because the computation of the normalizing constant is #P-hard, a complexity class harder than NP. Indeed, a solution of a NP-hard problem can be verified in polynomial time, while verifying the solution of a #P-hard problem is NP-hard.

A tractable alternative is the negative pseudo log-likelihood [Besag, 1975]:

$$\mathrm{NPLL}(S) = -\sum_{y\in S}\log(\prod_{i} P^{\mathcal{M}}(y_i|\mathbf{y}_{-i}))$$

where $\mathbf{y}_{-i}$ denotes the sequence $\mathbf{y}$ after removal of the value y_i.

The NPLL works at the level of each variable Y_i, in the context of $\mathbf{y}_{-i}$, the assignment of all other variables. In the case of Sudoku, it means we successively try to predict one cell knowing all the others. Computing each term $P(y_i)$ of the NPLL requires only normalization over one variable Y_i, a computationally easy task. Moreover, it is known to be asymptotically consistent, *i.e.,* it asymptotically converges towards the correct GM [Besag, 1975; Geman and Graffigne, 1986].

Gradients of the Negative Pseudo Log-likelihood

We now consider the case where the MRF $\mathcal{M}$ is the output of a neural network. This neural net can be trained using the NPLL as a loss, which requires to compute the NPLL gradients.

We assume $\mathcal{M}$ to be pairwise, *i.e.*, with cost function of arity at most 2. The cost function on a pair of variables (Y_i, Y_j) is denoted $\mathcal{M}[i,j]$. It is interesting to look into the contribution of every sample $\mathbf{y}$ to the gradient $\frac{\partial NPLL}{\partial \mathcal{M}[i,j](v_i,v_j)}$ of the NPLL for a given pair of values (v_i, v_j) of (Y_i, Y_j).

As proven in Annex A, the contribution of sample $(\omega, \mathbf{y})$ to $\frac{\partial NPLL}{\partial \mathcal{M}[i,j](v_i,v_j)}$ is:

$$\frac{\partial NPLL}{\partial \mathcal{M}[i,j](v_i,v_j)} = [\mathbb{1}(y_i = v_i, y_j = v_j) - P^{N(\omega)}(v_i|\mathbf{y}_{-i})\mathbb{1}(y_j = v_j)] + [\mathbb{1}(y_i = v_i, y_j = v_j) - P^{N(\omega)}(v_j|\mathbf{y}_{-j})\mathbb{1}(y_i = v_i)]$$

Bibliography

Allouche, D., De Givry, S., Katsirelos, G., Schiex, T., and Zytnicki, M. (2015). Anytime hybrid best-first search with tree decomposition for weighted csp. In *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*, pages 12–29. Springer.

AlQuraishi, M. (2019). Alphafold at CASP13. *Bioinformatics*, 35(22):4862–4865.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410.

Anand, N. and Achim, T. (2022). Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*.

Anfinsen, C. B. (1973). Principles that govern the folding of protein chains. *Science*, 181(4096):223–230.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR.

Baptiste, P., Le Pape, C., and Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media.

Baxter, J. (2000). A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198.

Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. (2000). The protein data bank. *Nucleic Acids Res.*, 28:235–242.

Besag, J. (1975). Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 24(3):179–195.

Bhattacharya, N., Thomas, N., Rao, R., Daupras, J., Koo, P., Baker, D., Song, Y. S., and Ovchinnikov, S. (2020). Single layers of attention suffice to predict protein contacts. *bioRxiv*.

Bhella, D. (2019). Cryo-electron microscopy: an introduction to the technique, and considerations when working to establish a national facility. *Biophysical reviews*, 11(4):515–519.

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

Bottou, L. and Bousquet, O. (2007). The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20.

Branden, C. I. and Tooze, J. (2012). *Introduction to protein structure*. Garland Science.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

Brouard, C., de Givry, S., and Schiex, T. (2020). Pushing data into CP models using graphical model learning and solving. In *International Conference on Principles and Practice of Constraint Programming*, pages 811–827. Springer.

Caruana, R. (1998). *Multitask learning*. Springer.

Chandonia, J.-M., Fox, N. K., and Brenner, S. E. (2019). Scope: classification of large macromolecular structures in the structural classification of proteins—extended database. *Nucleic acids research*, 47(D1):D475–D481.

Cooper, M., de Givry, S., and Schiex, T. (2020). Graphical models: queries, complexity, algorithms. *Leibniz International Proceedings in Informatics*, 154:4–1.

Croitoru, F.-A., Hondru, V., Ionescu, R. T., and Shah, M. (2023). Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

de Oliveira, S. and Deane, C. (2017). Co-evolution techniques are reshaping the way we do structural bioinformatics. *F1000Research*, 6.

Dechter, R. (2013). Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191.

Demoen, B. and de la Banda, M. G. (2014). Redundant sudoku rules. *Theory and Practice of Logic Programming*, 14(3):363–377.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Doersch, C. and Zisserman, A. (2017). Multi-task self-supervised visual learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2051–2060.

Dunbrack Jr, R. L. and Cohen, F. E. (1997). Bayesian statistical analysis of protein side-chain rotamer preferences. *Protein Science*, 6(8):1661–1681.

Dunker, A., Lawson, J., Brown, C. J., Williams, R. M., Romero, P., Oh, J. S., Oldfield, C. J., Campen, A. M., Ratliff, C. M., Hipps, K. W., Ausio, J., Nissen, M. S., Reeves, R., Kang, C., Kissinger, C. R., Bailey, R. W., Griswold, M. D., Chiu, W., Garner, E. C., and Obradovic, Z. (2001). Intrinsically disordered protein. *Journal of Molecular Graphics and Modelling*, 19(1):26–59.

Durante, V., Katsirelos, G., and Schiex, T. (2022). Efficient low rank convex bounds for pairwise discrete Graphical Models. In *Thirty-ninth International Conference on Machine Learning*.

Geman, S. and Graffigne, C. (1986). Markov random field image models and their applications to computer vision. In *Proceedings of the international congress of mathematicians*, volume 1, page 2. Berkeley, CA.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 3 of *Neural information processing systems foundation*, page 2672–2680.

Hadley, C. and Jones, D. T. (1999). A systematic comparison of protein structure classifications: Scop, cath and fssp. *Structure*, 7(9):1099–1112.

Hallen, M. A. and Donald, B. R. (2019). Protein design by provable algorithms. *Communications of the ACM*, 62:76–84.

Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *IJCAI (1)*, pages 607–615.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In press, I., editor, *A Field Guide to Dynamical Recurrent Networks, IEEE*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hu, Y., You, H., Wang, Z., Wang, Z., Zhou, E., and Gao, Y. (2021). Graph-mlp: Node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*.

Hurley, B., O'sullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., and Givry, S. d. (2016). Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, 21:413–434.

Jumper, J., Evans, R., Pritzel, A., and et al. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Kapoor, S. and Narayanan, A. (2022). Leakage and the reproducibility crisis in ml-based science. *arXiv preprint arXiv:2207.07048*.

Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, pages 4171–4186.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations - ICLR, San Diego - Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1188–1196, Bejing, China. PMLR.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

Levinthal, C. (1969). How to fold graciously. *Mossbauer spectroscopy in biological systems*, 67:22–24.

Li, S. Z. (2009). *Markov random field modeling in image analysis*. Springer Science & Business Media.

Liu, H., Dai, Z., So, D., and Le, Q. V. (2021). Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34:9204–9215.

Liu, X., Faes, L., Kale, A. U., Wagner, S. K., Fu, D. J., Bruynseels, A., Mahendiran, T., Moraes, G., Shamdas, M., Kern, C., et al. (2019). A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis. *The lancet digital health*, 1(6):e271–e297.

Matsugu, M., Mori, K., Mitari, Y., and Kaneda, Y. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Morcos, F., Pagnani, A., Lunt, B., Bertolino, A., Marks, D. S., Sander, C., Zecchina, R., Onuchic, J. N., Hwa, T., and Weigt, M. (2011). Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609.

Palm, R., Paquet, U., and Winther, O. (2018). Recurrent relational networks. *Advances in neural information processing systems*, 31.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Pearl, J. (1982). Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-82)*, pages 133–136. AAAI Press.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Ruffini, M., Vucinic, J., de Givry, S., Katsirelos, G., Barbe, S., and Schiex, T. (2019). Guaranteed diversity & quality for the weighted csp. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pages 18–25. IEEE.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48.

Sillitoe, I., Lewis, T., Cuff, A., Das, S., Ashford, P., Dawson, N., Furnham, N., Laskowski, R., Lee, D., Lees, J., et al. (2015). Cath: protein structure classification database. *Nucleic Acids Res*, 43(D1):D376–D381.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Taskar, B., Chatalbashev, V., and Koller, D. (2004). Learning associative markov networks. In *Proceedings of the twenty-first international conference on Machine learning*, page 102.

Taylor, W. R. (1986). The classification of amino acid conservation. *Journal of theoretical Biology*, 119(2):205–218.

Tsai, C.-J., Del Sol, A., and Nussinov, R. (2009). Protein allostery, signal transmission and dynamics: a classification scheme of allosteric mechanisms. *Molecular Biosystems*, 5(3):207–216.

Uhler, C. (2017). Gaussian graphical models: An algebraic and geometric perspective. *arXiv preprint arXiv:1707.04345*.

Van Beek, P. and Chen, X. (1999). Cplan: A constraint programming approach to planning. In *AAAI/IAAI*, pages 585–590.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *1st Conference on Neural Information Processing Systems (NIPS 2017)*.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., et al. (2017). Graph attention networks. *stat*, 1050(20):10–48550.

Vylomova, E., Rimell, L., Cohn, T., and Baldwin, T. (2016). Take and took, gaggle and goose, book and read: Evaluating the utility of vector differences for lexical relation learning.

Walian, P., Cross, T. A., and Jap, B. K. (2004). Structural genomics of membrane proteins. *Genome biology*, 5:1–8.

Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., et al. (2022). Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models. *bioRxiv*, pages 2022–12.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

Chapter 2

Computational Protein Design

The goal of this thesis is to propose a novel method for Computational Protein Design (CPD), based on both discrete optimization and deep learning. This chapter presents the context of this work.

First, Section 2.1.2 gives a brief overview of existing protein design methods, together with some successes and challenges. Second, Section 2.2.4 focuses on energy-based methods, the traditional approach for Computational Protein Design and the basis of this work. Third, Section 2.3.3 gives a comprehensive review of the recent deep-learning-based methods for CPD. They are sorted based on their input (either a protein structure or sequence), further classified into the protein representation chosen. Finally, we conclude on the respective advantages of both DL-based and energy based methods, and we present the novel hybrid approach we developed.

2.1 Generalities on Protein Design

This section provides an overview of existing design methods. It first presents the motivation through the wide variety of potential application areas, then describe existing methods, both experimental and rational. Finally, past successes and present challenges are highlighted.

2.1.1 Goal and Interest

Protein design aims to conceive new proteins with optimized or new functions or properties. While natural proteins may sometimes be suitable for certain applications, frequently, there is no existing protein that meets all the desired criteria. Therefore, custom-designed proteins hold significant importance across various fields. Protein design can involve the redesign of an existing protein or the *de novo* design of a new protein from scratch.

Protein design has seen notable successes across a wide range of application fields and protein types. Antibodies, for instance, are of great interest in medicine, both for therapy and diagnosis. Previous accomplishments include the design of antibodies with significantly enhanced binding affinity to target proteins [Clark et al., 2006] and the development of a protein-based diagnostic tool [Yeh et al., 2023a]. Enzymes, with their catalytic capabilities, play crucial roles in green chemistry and biotechnology for sustainable bioproduction and biotransformation processes. They are used in various industrial processes, such as starch processing with amylases and biomass conversion with cellulases. In order to make enzymes fit for industrial processes, design can aim to enhance their thermostability [Lu et al., 2022], to make them able to catalyze new reactions [Röthlisberger et al., 2008; Jiang et al., 2008] or to use a more cost-effective cofactor [Mallinson et al., 2023]. Additionally, the design of protein assemblies has significant applications in nano(bio)technologies, including the utilization of molecular machines [Ng et al., 2019] and self-assembling proteins [Noguchi et al., 2019; Votteler et al., 2016].

Protein engineering has thus become a key technology for tailoring protein functions and properties to meet specific requirements. Design strategies can be split into two non-exclusive categories described below: experimental and rational approaches.

2.1.2 Overview of Existing Methods

Experimental Approaches

Experimental approaches are based on directed evolution [Packer and Liu, 2015; Romero and Arnold, 2009], whose application on enzymes was crowned in 2018 by Frances Arnold's Nobel prize. The main idea is to modify genetic information by random mutagenesis or gene shuffling in order to mimic natural evolution. It is based

on two iterative steps: the generation of libraries of mutants, and high-throughput screening to select mutants with high activity.

This method is efficient to alter or optimize existing proteins, and it does not require any knowledge on the structure or the ability to predict the effect of a mutation. However, it is costly, both in time and in resources, and the required high-throughput screening is not available on all design targets. Moreover, only a limited fraction of the sequence space is explored.

Rational Approaches

Rational approaches exploit knowledge on structure and/or function (which is not always available) to make the desired changes.

Traditional rational design approaches are based on bioinformatics (phylogeny analysis, ancestral sequence reconstruction, ...) and molecular modelling (3D structural model prediction, docking, molecular dynamics, analysis of intra/inter molecular interactions, ...). For instance, when an active site is identified (possibly via the study of conserved positions in homologuous proteins or through molecular modelling), point mutations can be tested for increased activity [Shlyk-Kerner et al., 2006]. Those approaches require knowledge and expertise on the target protein, and the exploration of the sequence space is limited to a very small number of variants, mostly single-point mutations. They can be combined with experimental approaches such as site-directed mutagenesis —to make specific mutating change in DNA— or controlled randomization to guide the construction of small-size library where the diversity is focused on key regions.

More recently, machine learning methods have been leveraged to produce fitness model of proteins, *i.e.*, predicting a property from the sequence. Such models can then be used to select the most promising mutants for the target property. Two strategies exist to estimate those models, and they can be combined [Hsu et al., 2021]. First, a ML model can be fit on a dataset of protein mutants with properties assessed in lab, for instance via mutational scanning experiment. Experimentally-labelled data are difficult to generate, thus they are not always available. Even when they are, the amount of data is limited and they are very often restricted to one or very few mutations. Second, a ML-based fitness model can be estimated from evolutionary data. Much information can be derived from a Multiple Sequence Alignment (MSA) of homologuous proteins (as detailed in Subsection 1.1.3), including per-position preferences and co-evolving residues. This can be used to estimate a Hidden Markov Model or a Markov Random Field (the terminology Potts model is also used) [Vorberg et al., 2018] describing the fitness landscape. Recently, the coupling a DL-based fitness estimator with a sampling procedure resulted in a β-lactamase, an enzyme associated with anti-bio-resistance, with both higher stability and activity [Fram et al., 2023].

In terms of automated *in silico* exploration and filtering of the whole sequence space, the most successful approach is *Computational Protein Design* (CPD). It is defined as the computer-aided rational design of a protein that folds into a structure to facilitate a function or property [Samish, 2017]. The most usual approach to CPD consists in choosing or *de novo* constructing a target backbone that could carry the function of interest, and then identify a sequence that will fold onto this backbone and present the expected properties. In this case, the input of the problem is the target backbone, and the output is the designed sequence(s). This approach is sometimes referred as the *inverse folding problem*.

Pioneering works focused on redesigning the hydrophobic core of existing proteins based on knowledge on protein stability [Dahiyat and Mayo, 1996]. As methodological advances were achieved, design objectives diversified and targets expanded. In 2003, a new topology fold, named top7, was designed from scratch, paving the way toward *de novo* protein design [Kuhlman et al., 2003]. New functional enzymes [Jiang et al., 2008], and enzymes with new substrate specificity [Verges et al., 2015] are other remarkable achievements. The size and complexity of design keeps increasing with recent self-assembling nanocages [Votteler et al., 2016] and tunable protein biosensors [Quijano-Rubio et al., 2021].

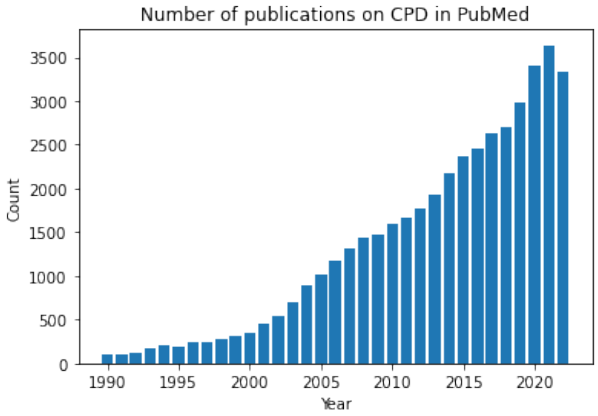


Figure 2.1: Number of publications on Computational Protein Design in the PubMed database since 1990.

All those successes may explain why Computational Protein Design is a rapidly-developing area, as illustrated in Figure 2.1. The methodological developments behind those successes are described in the next subsection.

2.2 Energy-based Methods

Computational Protein Design (CPD) aims at finding the best-suited sequence(s) folding onto an input 3D protein structure so that it fulfills a function of interest. The CPD problem is formulated as an optimization problem: it aims to find the most stable protein sequence on the input structure. Since maximizing stability is equivalent to minimizing energy, sequences are selected based on an objective function quantifying, among others, their energy.

The CPD framework requires 3 ingredients. First, the **designed system** (represented by the input backbone) must be realistically modelled to carry the function or properties of interest. This work will not detail how it is obtained. Second, the **objective function**, and particularly the energy function, should be defined as accurately as possible. Third, an efficient **optimization framework** is needed.

This section first introduces the paradigm of CPD and highlights the challenges to face, then it describes the existing energy functions and it finally details the existing optimization methods.

2.2.1 Paradigm and Challenges

Computational Protein Design is a problem whose input is a protein 3D structure backbone, and the output is a sequence (or a library of sequences) likely to fold onto this backbone. An energy-based objective function is chosen to score sequences, then the sequence space is explored to obtain the designed sequences. They are to be tested experimentally, and the result can be integrated to the protocol via a feedback loop. The complete CPD protocol is summarized in Figure 2.2.

The main challenge faced by CPD is the combinatorial explosion of the search space. Indeed, at each position of the sequence, one amino acid among the 20 canonical ones must be chosen, resulting in 20^n possible sequences, with n the length of the sequence. For a small protein of 100 residues, it represents about 10^{130} possible sequences, which is more than the number of atom in the universe. The search space is even larger if in addition to the choice of amino acid, its side chain must be placed properly in 3D space.

In order to restrain the astronomical search space, simplifying hypotheses are usually used. First, the backbone of the protein is assumed to be rigid. Second, the side chain of residues can only adopt discrete conformations called rotamers. Libraries of the most common rotamers, such as Dunbrack's, are used in practice [Dunbrack Jr, 2002]. Nevertheless, despite these simplifications, CPD remains a non-deterministic polynomial-time hard (NP-hard) problem [Pierce and Winfree, 2002].

Under those assumptions, the main objective of design can be formulating as

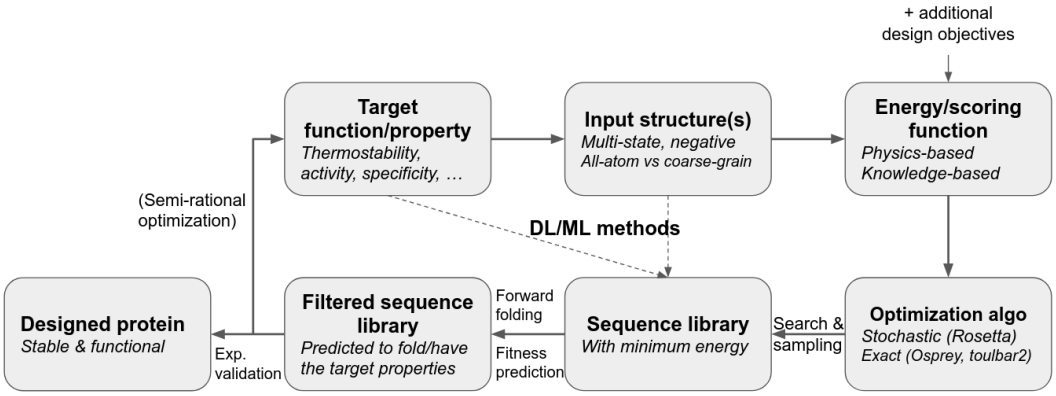


Figure 2.2: The protocol of Computational Protein Design. From the design objective, an (or several) input 3D backbone is crafted. An energy function is used to score sequences on the input structure, then the sequence space is explored to find low-energy sequence(s). The designed sequences are then post-evaluated *in silico* and validated experimentally. Optionally, they can be subsequently optimized by a round of directed evolution, or experimental information can be used to guide computational design. Deep-Learning based methods simplify this protocol (dotted lines) by directly mapping the objective or the input structure to a sequence.

finding the sequence s minimizing the energy E:

$$s^* = \underset{s \in S}{\operatorname{argmin}} \min_{r \in R(s)} E(r)$$

S in the sequence space and $R(s)$ is the ensemble of rotamers associated with a sequence s. The optimal conformation (sequence + rotamers) s^* for the input backbone is called Global Minimum Energy Conformation (GMEC). The energy E is given by an energy function, that scores each of the possible conformations. They are detailed in Subsection 2.2.2.

For computational efficiency, a simplified energy is often used. It is written as a sum of unary (one-body) and pairwise (two-body) terms, with i and j representing 2 positions in the sequence:

$$E(r) = E_\varnothing + \sum_{i=1}^{n} E_i(r_i) + \sum_{i<j} E_{i,j}(r_i, r_j)$$

Those terms represent interactions between residues in the protein, but also possibly with their partners (ligand, cofactor, other proteins, ...).

In the rest of this chapter, we will focus on the optimization of the energy. However, it is important to note that for practical design, maximizing stability

is rarely the only objective. Indeed, to design a specific function and required properties, specific global and local biochemical and geometrical constraints are imposed. Possible additional objectives include symmetry of the designed sequence, diversity, high binding affinity to a ligand, or requirement in terms of composition (for instance, prohibit cysteins to avoid the formation of disulfide bridges).

2.2.2 Energy and Scoring Functions

Energy functions aim to model physical interactions between atoms. They need to be accurate, but also computationally tractable. Applying full first principle quantum chemistry calculations on macromolecular systems such as proteins is not feasible. Therefore, approximations have been developed in the form of physics-based force-fields used for molecular dynamics. As protein design requires a trade-off favouring more computational efficiency over accuracy, physics-based energy terms have been simplified and enriched with statistical terms to form scoring functions.

Physics-based Energy

Molecular mechanics energy functions, or force fields, are based on the principles of Newton's classical physics. The molecular system is described at the atomic level, and the interactions within are modelled by the force field, used to compute the potential energy of the system [Ponder and Case, 2003].

The total potential energy of a system is calculated as the sum of energies representing bonded and non-bonded interactions between atoms.

$$E_{total} = E_{bonded} + E_{non-bonded}$$

Bonded terms correspond to interactions between covalently-bonded atoms. They include a bond-stretching term, and angle-bending term and a dihedral-rotation term. Non-bonded interactions are the sum of electrostatic and van der Waals interactions, modelled as Coulomb and Lennard-Jones potentials. Coulomb potential results from the attractive or repulsive force between charged particles, while Lennard-Jones potential models the fact that two interacting particles repel each other at very close distance, attract each other at moderate distance, and do not interact at infinite distance.

The molecular system may include one or several proteins as well as small molecules. Therefore, the interactions represented include inter and intra protein interactions and binding interactions with a ligand. In addition to interactions inside the molecular system, interactions with the solvent need to be represented. Indeed, for the majority of proteins that folds into aqueous environment, it is the hydrophobic interactions with water that drive the burial of the hydrophobic core. Membrane proteins are in a lipidic environment and thus interact differently with their solvent.

The solvent can be represented either explicitly as a set of water molecules, which is computationally expensive, or implicitly by a dedicated energy term (a dielectric constant).

Force fields are widely used for molecular dynamics simulations. Among the most commonly used for proteins, there are AMBER [Maier et al., 2015] and CHARMM [Vanommeslaeghe et al., 2010]. They are usually combined with solvent models computationally too costly to be used for design.

Scoring Functions

Force fields do not take into account some features observed in protein structures. Therefore, some additional terms, statistically derived from experimental observations, have been introduced.

For instance, hydrophobic interactions tend to result in hydrophobic residues being buried at the core of proteins, and polar residues being exposed at the surface. However, polar residue can be important at the core of the protein. The introduction of a hydrogen-bonding rule for the introduction of polar or charged residues in the core resulted in an improved stability or thioredoxin core design [Bolon et al., 2003].

A good illustration of the mix between physics-based and statistical terms is the Rosetta energy function widely used for design named ref2015 [Alford et al., 2017].

$$E_{\text{total}} = \lambda_1 E_{\text{non-bounded}} + \lambda_2 E_{\text{solvent}} + \lambda_3 E_{\text{torsion}} + \lambda_4 E_{\text{bonded}} + \lambda_5 E_{\text{ref}}$$

- The $E_{\text{non-bounded}}$ term represents physical interactions as previously (Coulomb and Lennard-Jones potentials, and hydrogen bonds).
- The E_{solvant} term represents the solvent. Indeed, if an energy scoring function needs to represent the solvent, computationally-expensive explicit representation are seldom used in design.
- The torsion terms E_{torsion} include rotamer and dihedral preferences.
- Finally, the reference energy term E_{ref} aims to account for the variable chemical composition of amino acids.

The importance of each energy term is weighted by factors λ_i that are derived from a training set containing both small and macro-molecules. They are crucial since the drawback of combining physics and statistics is that some terms can be partially redundant.

When combining physics-based and statistical terms, the resulting function does not represent potential energy anymore. Therefore, it is called a *scoring function.*

Nevertheless, it is aimed to be used the same way. In the case of design, the scoring function is used to guide the conformation-sequence search and estimate the likelihood of a conformation.

The conformation-sequence space contains all the possible sequences and the possible placements of the side chains, represented by rotamers. Since the main challenge of protein design comes from the huge size of this search space, reducing it will likely increase resolution time. Therefore, one way to increase the efficiency of scoring function is *coarse-graining*: instead of representing all the atoms of the protein, a residue-level representation is used. The design problem reduces to finding the residue without explicit consideration of side chain placement.

Knowledge-based Potentials

In order to further improve the speed/accuracy balance of scoring functions, some approaches completely replaced all the physics-based terms by statistical terms. Such approaches are called knowledge-based potentials (KBP) [Poole and Ranganathan, 2006]. Potential is a terminology from physics used to qualify any term defining a potential energy function.

Most KBPs are based on the inverse Boltzmann equation:

$$E_{i,j} = -RT \ln(\frac{P^{obs}(i,j)}{P^{ref}(i,j)})$$

with T the temperature, R the gas constant and (i,j) the current pair of atoms or residues. P^{obs} is the joint probability of observing the pair of residues (i,j) while P^{ref} is the reference probability. One classical reference state is simply the average over the 20 different amino acid types [Samudrala and Moult, 1998].

These probability distributions are computed on known structure from the PDB. At first, they were based solely on contacts via a distance cut-off [Tanaka and Scheraga, 1976; Park and Levitt, 1996]. Improvements have been made through the consideration of distance-dependency [Zhou and Zhou, 2002], then the inclusion of orientation terms [Buchete et al., 2004]. Current KBPs differ from each other in the way they consider the distance and the orientation preferences of the pairwise contacts [Zhou and Skolnick, 2011; Lu and Skolnick, 2001; López-Blanco and Chacón, 2019].

More recently, deep learning has been used to learn protein-specific potential. Those potentials present the advantages of not requiring pairwise decomposition, and they are computationally efficient. They have been used for conformation prediction [Du et al., 2020a], to generate designable backbone [Huang et al., 2022] and for unsupervised contact prediction [Sercu et al., 2021]. The neural net for protein structure prediction AlphaFold [Jumper et al., 2021] shows evidence of having learned something akin to an energy function for scoring the accuracy of candidate

structure [Roney and Ovchinnikov, 2022]. For design, the energy has been modelled as the log-probability of the sequence conditioned by the structure, and candidate sequences sampled accordingly [Anand et al., 2022].

2.2.3 Existing Algorithms

CPD requires to solve NP-hard problems on exponential search space [Pierce and Winfree, 2002]. In practice, for the sizes of the problems that need to be solved, solutions can be produced. Existing approaches rely either on stochastic optimization or provable methods.

Stochastic Approaches

The widely-used for design Rosetta Molecular Modelling Suite [Leaver-Fay et al., 2011] relies on Monte-Carlo Simulated Annealing [Van Laarhoven et al., 1987] to produce low-energy sequences.

Starting from a random sequence, the sequence space is explored locally by assessing a random mutation at each iteration. It is accepted based on the Metropolis criterion. If the energy of the new sequence decreases, the mutation is accepted. If the energy increases, the mutation has a probability of acceptance proportional to the negative energy difference and inversely proportional to the system temperature. The idea behind the Metropolis criterion is to be able to overcome energetic barriers and thus avoid to be stuck in a local minimum. The higher the temperature, the less selective the criterion and thus the more diverse the sequences. Usually, iterations start at a high temperature to sample space, then the temperature decreases progressively toward a minimum.

The Metropolis criterion is stochastic. Therefore, even when starting from the same initial sequence, a different sequence is obtained at the end of the procedure, with no guarantee of reaching the global optimum (even when it is reached, there is no way to know it). Therefore, the design procedure is often repeated several times to generate a library of low-energy sequences.

First Exact Methods

Deterministic approaches do not involve any random process, and they build a proof that the obtained solution is optimal (or close to the optimum). To do so, they explore a search tree. Each of its node is a sub-problem defined on on a restricted search space. A solution is found when a leaf node is reached.

A well-known protein design software based on an exact method is Osprey [Hallen et al., 2018]. It identifies the GMEC using a combination of Dead-End-Elimination (DEE) [Desmet et al., 1992] and A* [Hart et al., 1968] (a best-first search Branch & Bound). DEE reduces the search space by eliminating energetically-dominated

rotamers, as they cannot be part of the optimal solution. A* explores the remaining search tree to find the lowest energy solution.

This algorithm suffers from exponential space and time complexities, which limits its application to small proteins. More efficient tree search algorithms exist. In particular, the formulation of the CPD problem as a Cost Function Network (CFN) and its resolution with standard depth-first search algorithm presents a polynomially bounded space complexity. Moreover, the CFN-based approach has been shown to solve the CPD problem faster than the DEE/A* approach [Allouche et al., 2014] and it is empirically efficient when combined with Rosetta scoring functions [Simoncini et al., 2015].

CPD with Cost Function Networks

The CPD problem can be expressed as a Cost Function Network (X, D, C) [Traoré et al., 2013].

- Each position in the sequence is associated with one variable X_i. If the sequence is composed of n amino acid, $i \in \{1, \ldots, n\}$.
- The domain of a variable is either the 20 canonical amino-acid (*coarse-grain* case) or a set of discrete rotamers (all-atom case). In the coarse-grain case, only the amino acid identity is to be chosen; in the all-atom case, the conformation of the side chain must be decided too. This work uses the coarse-grained representation, and therefore D_i is of size 20 for all variables X_i.
- The cost functions C correspond to the unary and binary energy terms E_i and $E_{i,j}$. The binary terms are represented as a matrix of shape 20×20 indicating the cost of assigning each pair of variables.

Once expressed as a CFN, the CPD problem is solved by toulbar2 which finds the optimal sequence for the input backbone. In addition to the optimal solution, toulbar2 is also able to exhaustively enumerate all the sequences within a bounded energy gap, which is useful to generate libraries of sequences to be tested experimentally. This functionality was further improved to generate sequence libraries with guaranteed diversity [Ruffini et al., 2019].

Applications sometimes target functions that cannot be represented by a single input backbone. For instance, for a protein designed to bind to a given target, the designed sequence should fit both the bound and unbound conformations. In other cases, the designed protein should be able to recognize several ligands or to adopt different conformation to fulfill its function. In all those case, it is useful to consider 2 (or more) input structures, and optimize the sequence on both structures. This can be done via positive multi-state design (MSD) [Vucinic et al., 2020]. In other cases, the designed protein should assembles or recognize some partners, but not

others. This can be expressed as favouring some conformations, and disfavouring others, which can be done by negative design.

2.2.4 Limitations

This formulation of CPD has known many successes (some of them have been detailed in Subsection 2.1.2) but it suffers from some limitations. To begin with, the problem is ill-defined: it targets the minimum-energy sequence for the input structure, but the input backbone is not necessarily the structure of minimum energy for the sequence, which implies the designed sequence will not necessarily fold onto the target structure.

In addition, each of the 3 ingredients restricts the method. Representing the target protein as a 3D backbone requires some prior knowledge on the protein structure. Moreover, this backbone is usually considered as rigid and thus it does not account for the flexibility of the protein, which is often crucial for its function. It is only partially alleviated by multi-state design.

Second, the quality of the designed sequence directly depends on the quality of the objective function. However, regardless of the energy function chosen, it is an approximation, which can degrade the quality of solutions. Third, the optimization problem is NP-hard. Therefore, even the most efficient algorithms become intractable as the size of the problem (*i.e.,* the number of residues on the protein) increases.

These limitations may explain the shift actually occurring toward deep-learning based methods. By learning a direct mapping from structure or function to sequence, they bypass the difficulties raised by the optimization of an energy function.

2.3 Protein Design with Deep Learning

In the wake of its tremendous successes in processing image video and speech, Deep Learning (DL) has been applied to many fields, including structural biology. In this area, the most visible success has certainly be AlphaFold for protein structure prediction [Jumper et al., 2021]. It demonstrates the ability of neural nets to learn the complex relationship between protein sequences and structures. This success was made possible by the existence of the Protein Data Bank (PDB) [Berman et al., 2000], which gathers all the protein structures experimentally resolved and makes them publicly available.

In the wake of this breakthrough, DL has been applied to many prediction tasks involving protein structures, including protein-protein interaction [Gainza et al., 2020], protein binding interfaces [Krapp et al., 2023], protein flexibility [Vander Meersche et al., 2021] and protein conformation [Cretin et al., 2021]. DL has also become a very appealing approach for Computational Protein Design (CPD). The first two subsections present the DL existing approaches for structure-based and sequence-based design. They follow and update a review paper we published in 2021 [Defresne et al., 2021]. Then, we discuss how to assess and compare DL-based CPD methods.

Figure 2.3 summarizes the classification between DL-based protein design methods presented in this section. It relies on a first level on the distinction between structure-based and sequence-based methods. On a second level, methods are classified based on the type of representation they use.

2.3.1 Structure-based Methods

Applying DL to CPD is not straightforward. Indeed, DL methods require suitable representations of both input and output data. In the case of CPD, the output is a protein sequence, whose length vary between instances. The input is a 3D structure, which contains a rich relational information that should be concisely encoded. Indeed, protein structures are naturally insensitive to rotations and translations. Ideally, the chosen representation should account for this property (referred to as rotation/translation *invariance*) to make training efficient.

This section is organized around the different protein structure representations proposed. The first ones were based on hand-crafted features, which was convenient to use but did not leverage geometrical properties of the structure. To date, four major types of representations are used: voxels, distance maps, graphs and point clouds (which are a degenerated case of graph). We describe the use, benefits and drawbacks of each for CPD, as well as the neural architectures used to process them. All the approaches described are summarized in Table 2.1.

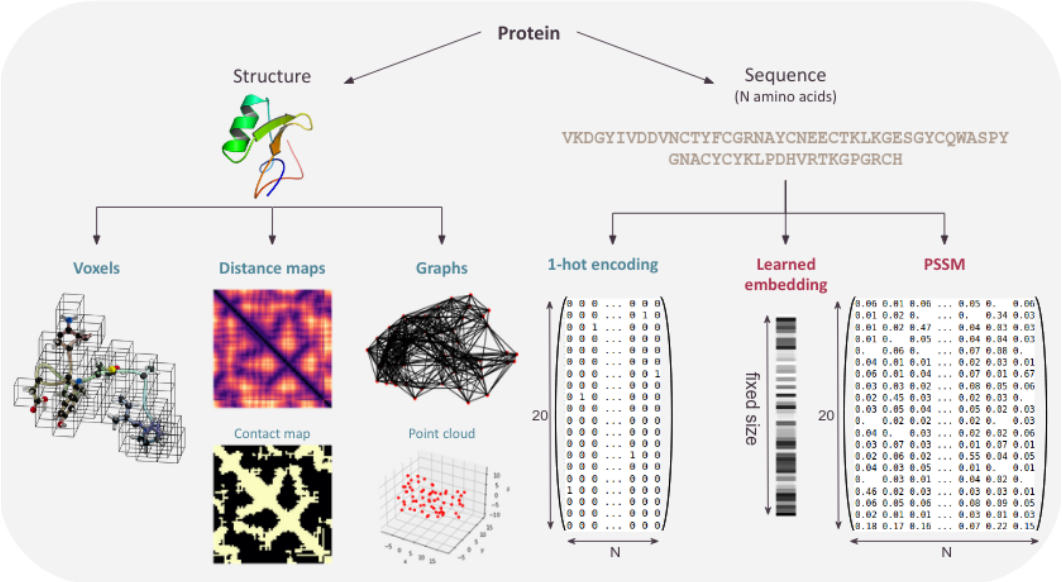


Figure 2.3: Protein structure and sequence representations used for deep learning. The sequence can be both an input and an output: language models process a one-hot encoding into a learned embedding, while structure-based methods often produce a PSSM from which designed sequences are sampled. Among the structure representations, the graph-based are currently the more popular, but there is still no consensus.

	Hand-crafted	Voxels	Distance map	Graph	Point clouds
MLP	[Li et al., 2014; Wang et al., 2018; O'Connell et al., 2018]	-	-	-	-
CNN	-	[Shroff et al., 2019; Zhang et al., 2020; Qi and Zhang, 2020; Lu et al., 2022; Anand et al., 2022]	[Norn et al., 2020; Anishchenko et al., 2020; Chen et al., 2020; Tischer et al., 2020; Wicky et al., 2022; Syrlybaeva and Strauch, 2023]	-	-
GNN	-	-	-	[Strokach et al., 2020; Jing et al., 2021; Dauparas et al., 2022; Hsu et al., 2022; Gao et al., 2023]	[Eliasof et al., 2021]
Transformer	-	[Liu et al., 2022; Huang et al., 2023]	-	[Ingraham et al., 2019]	[Du et al., 2020b]
Diffusion	-	-	[Lee et al., 2022]	-	[Ingraham et al., 2022; Anand and Achim, 2022; Watson et al., 2023]

Table 2.1: Structure-based design methods described in this manuscript, organized by the type of structural representation, and the neural architecture it is fed into. It is worthy to note diffusion is not a type of architecture but a training scheme; nevertheless, we singled it out to highlight the recent trend of its usage for protein design.

Hand-crafted and Sequential Representations

Given the input protein backbone, one straight-forward way to represent it as a tensor is to consider it as a sequence of distances, angles and dihedrals (as usual in molecular modelling). While rotation and translation invariant, this 1D representation is not ideal in the context of deep or machine learning because of its sensitivity to noise through the well-known "lever effect": a tiny change in one dihedral angle may translate in very large changes in distant Cartesian coordinates, resulting in drastic effect on energy [Li et al., 2014; Wang et al., 2018; O'Connell et al., 2018].

Instead, each position in the sequence can be associated with per-residue geometric and structural features (characteristics deemed to be significant for the learning problem), possibly completed with other features related for example to the target design properties. Such an approach was used in the first DL-based CPD systems, starting with SPIN [Li et al., 2014] and its extension SPIN2 [O'Connell et al., 2018], as well as work by [Wang et al., 2018]. They differed by the features considered, but all of them stacked the features and fed them into a MLP to produce a protein sequence by predicting the probability of each amino acid type at each position (which corresponds to a Position-Specific Scoring Matrix).

The main advantage of using hand-crafted structural features is to obtain a fixed-size feature vector, which is required by many DL architectures, including MLPs and CNNs. Even recurrent models, that accept sequences of variable length, need a fixed-size word embedding (the amino acid feature vector). The obvious drawback resides in the features themselves. Crafting and selecting suitable features, with enough information to learn the complex sequence/structure relationship, is a hard task and important information may just be lacking (or some may be redundant). Neural networks have an ability to automatically extract features that should be leveraged, but it requires to represent the entire structure instead of residue-level features.

Voxel Representation

As a three-dimensional object, a protein structure can be directly analyzed as volumetric data (like a 3D image) that can be fed into a 3D-CNN: the space is discretized into cubic voxels of typically 1 Å side. Atoms within each voxel are counted and a Gaussian filter is applied to the discrete count to produce an occupancy map. Each type of atoms is counted independently, and treated as different channels, like RGB channels [Anand et al., 2022; Qi and Zhang, 2020; Shroff et al., 2019; Zhang et al., 2020].

The usually considered task is to predict an amino acid given its local environment. The input of the network is a box of size about 20 Å, centered on the target residue. The geometrical environment is canonized to ensure rotational and translational invariance. This input, of shape $C\times$(volume of the cube) with C the

number of channels, is fed into a 3D CNN (whose architecture details vary between approaches), which is trained on existing proteins, to output a probability over the 20 possible amino acids. This task is also useful for single-point mutation prediction [Shroff et al., 2019, 2020], or for protein design, either indirectly (using the prediction to reduce the search space of Rosetta) or directly by taking the maximum probability for each amino acid [Zhang et al., 2020], or by sampling [Anand et al., 2022]. While DenseCPD [Qi and Zhang, 2020] outperformed several competitors in a recent benchmark experiment [Castorina et al., 2021], these results have to be taken with caution because the training and testing datasets used were not separated in the evaluation.

The main interest of this approach is the ability of 3D-CNNs to take into account the geometry of the protein structure. 3D-CNNs are used to identify structural motifs, independently of their scale or position, which is critical to decipher the sequence/structure relationship. The sensitivity of CNNs to rotation is cancelled by the use of canonized frames but this may not be always sufficient in the context of protein complexes [Eismann et al., 2021]. This limitation can be bypassed by rotational data-augmentation, which increases computation time. Discretization also requires to settle compromises between computational complexity and fidelity.

More recent approaches, ABACUS-R [Huang et al., 2023] and ProDESIGN-LE [Liu et al., 2022], have taken over the same task of predicting an amino acid from its environment but tried to overcome the limits of CNNs. First, they reason at the residue-level instead of the atom-level to avoid the computationally-heavy step of side chain optimization. Second, the environment is described by features on the nearest neighbours of the central residue, which are fed into a Transformer. The details of these features vary with the approach, but they are based on coordinates in a rotation and translation invariant fashion. The use of the nearest neighbours remove the need of a discretization of space, but it imposes the same number of neighbours for each environment. Like CNN-based approaches, the neural net outputs a probability over the type of amino acid, which is iteratively sampled to produce the sequence. Both approaches have been experimentally validated.

Distance Maps

A distance map is a 2D representation of a protein structure. It is a $n \times n$ matrix (n being the length of the protein) giving the distance between C_α atoms of each pair of amino acids. Contact maps are binary maps obtained by putting a distance threshold on the distance map. Both contact and distance maps have been massively used for protein structure prediction methods, including AlphaFold [Senior et al., 2020], trRosetta [Yang et al., 2020] and their successors [Jumper et al., 2021; Baek et al., 2021].

These structure prediction networks can be partially *inverted* to iteratively bias a sequence (starting from a random one) towards a target structure. The inversion

is based on so-called symbolic gradients: when an input sequence and an output structure are given, backpropagation can not only compute gradients on the weights (for training) but also on the input. The first structure prediction network to be inverted [Norn et al., 2020] was trRosetta, which predicts (among other things) a distance map from a sequence. The inversion process tends to optimize the input sequence so that it folds in the target structure. The major claimed advantage here is that the resulting sequence seems to avoid the pitfall of the usual ill-posed CPD problem: the predicted sequences seem to implicitly avoid the existence of alternate stable backbones. Soon after its release, AlphaFold [Jumper et al., 2021] was inverted in a similar way [Goverde et al., 2023] .

Structure prediction networks can also be exploited to hallucinate ideal proteins. The first proposed approach is iterative: a random single mutation is applied on a sequence, the distance map of the new sequence is computed using trRosetta [Anishchenko et al., 2020], and the mutation is kept based on an objective encouraging the sequence to be different from background. This process was iterated, starting from a random sequence. Around 20% of the designed sequences were experimentally shown to produce folds consistent with the predicted structure. However, the generated structure could not be controlled. This method was improved to generate symmetric protein assemblies [Wicky et al., 2022], with the long-term goal of designing complex components for nanomachines and biomaterials. A combination of AlphaFold hallucination and inpainting (*i.e.,* masking then predicting some part of the sequence) was used to successfully design *de novo* proteins with active sites and protein binding sites [Wang et al., 2022].

Distance map can also be used directly to represent an input structure. SPROF extended SPIN2 by incorporating 3D information in the form of a contact map [Chen et al., 2020]. The map is processed as an image, and the corresponding sequence as a caption. Then, taking inspiration from the usual "image captioning" task, they coupled a RNN and a CNN to output a PSSM, from which a designed sequence is produced. A similar idea was used to design protein-protein interactions [Syrlybaeva and Strauch, 2023].

As a 2D representation of a 3D structure, contact and distance maps offer several advantages. They are a low-dimensional, which makes computations efficient. They are images that benefit from all DL methods developed in the field such as CNNs with residual connections [Senior et al., 2020; Yang et al., 2020; Chen et al., 2020]. Finally, they are invariant for rotation or translation of the protein structure. The dimension reduction leads to the loss of some geometric information that can often be recovered [Adhikari and Cheng, 2018].

Graph Representations

Graphs are well-suited to represent relationships, here between residues in a protein structure. In the most basic graph, each node, or vertex, corresponds to one residue,

and edges connect pairs of residues within a distance threshold. Such a graph is equivalent to a contact map. A graph can be advantageous when there are few interactions between amino acid (modelled by a small distance threshold). With a sparse graph (with few edges compared to a complete graph), computations can be more efficient than on a distance map which explicitly represents all pairwise interactions. Contact maps are naturally sparse as the number of contacts of each residue is bounded. Moreover, additional information can be injected into so-called node and edges attributes. For instance, commonly used edge attributes include distances, direction and orientation between residues [Ingraham et al., 2019; Strokach et al., 2020; Dauparas et al., 2022] while dihedral features are sometimes added as node features. Table 2.2 summarizes the features considered in the main approaches.

Graph-based representations have gained in popularity since the pioneer work of the Structured Transformer proposed in 2019 [Ingraham et al., 2019]. This method considered direction and orientation from a local coordinates system, thus enabling rotation and translation invariance. A Transformer architecture was adapted and used as an encoder to process the graph. Then an auto-regressive decoder produces the output sequence, residue after residue. This work had been a basis to following approaches, which often reuse the same dataset for training and test, enabling a fair comparison between approaches. These methods differ mainly in the features considered and the neural architecture used to process them.

The Geometric Vector Perceptron (GVP) [Jing et al., 2021] used similar node and edge features (even though encoded differently), but used a Graph Neural Network instead, with a specific graph convolution that was rotation and translation equivariant. The resulting operations were also computationally more tractable than other equivariant approaches [Fuchs et al., 2020]. In parallel, the ProteinSolver [Strokach et al., 2020] also proposed their own implementation of message passing operations to propagate information between neighbouring residues. Like GVP, node and edge embedding are processed through several graph convolution and aggregation blocks. In both cases, the final node embeddings are fed into feed-forward layers to predict the missing residues auto-regressively.

Further improvement resulted from a change of features: instead of relative positions and orientations between neighbouring residues, ProteinMPNN [Dauparas et al., 2022] simply uses distances between all the backbone atoms, which provides a better inductive bias. The network was proposed by the Baker lab, which also developed the Rosetta tools for design, and it had been extensively experimentally validated, and proved to have higher success rates than the classical design tools. Improvement of a similar range was made possible by training a GVP-like architecture on tens of thousand of structures predicted by AlphaFold [Hsu et al., 2022]. Finally, PiFold [Gao et al., 2023] suppressed the auto-regressive decoding of the sequence and directly output the designed sequence, enabling a faster design. However, it also limits the possibility of controlling the design outputs, which is often

Approach	Edge attributes					Node attributes	
	C_α dist.	bb dist.	dir.	orient.	seq	dihedral	dir.
Structured Trf	✓		✓	✓	✓	✓	
Protein Solver	✓				✓		
GVP	✓		✓		✓	✓	✓
ProteinMPNN	✓	✓			✓		
PiFold	✓	✓	✓	✓	✓	✓	✓

Table 2.2: Edge and node attributes used in graph-based approaches. For edge attributes, distances (either between α carbons only or all atoms in the backbone), direction (dir) and orientation (orient.) are between a pair of residues. The attribute seq refers to the number of residues between in the sequence. For node attributes, the direction (dir.) can be the direction toward the side chain, or with the previous/following residue [Jing et al., 2021]. The way these features are encoded may vary between approaches.

necessary in practice (see Subsection 2.3.3 for details on how to assess DL-based design methods).

Like any approach using a coarse-grained representation (*i.e.,* at the residue level instead of the atomic level), difficulties linked with the side chain flexibility are elegantly side-stepped. However, it makes it impossible to represent molecules which are not made of amino acids, which is the case of many ligands. An all-atom graph was used to learn a protein structure representation that takes into account the primary, secondary and tertiary level of a protein [Hermosilla Casajús et al., 2021]. The pooling operations on the graph resulted into a change of scale, from all-atom to residue level then to even coarser scale. However, to our knowledge, all-atom graph representations have yet to be used for protein design.

Point Clouds (3D Coordinates)

The most brutal way to represent a structure is probably as a point cloud, the list of all 3D coordinates of its constituents, much like a PDB file. This dense information can be filtered to just keep the coordinates of C_α atoms [Eliasof et al., 2021], or an all (heavy) atom representation can be preserved [Du et al., 2020b].

Such representations have been used by MimNet [Eliasof et al., 2021], a GNN-like architecture that is reversible and therefore can do both folding (forward) and design (backward). Simultaneously training on both tasks results into a better design when the structure is better predicted. Coordinates can also be considered as a set of points, as in Atom Transformer, a network learning an energy function to predict the protein conformation [Du et al., 2020b].

By comparison with the general graph representation, point clouds do not reduce

the geometric information about the structure. They can be directly processed by rotation-equivariant operations, avoiding data augmentation. However, these operations are often costly in terms of computation time [Fuchs et al., 2020].

Very recently, point cloud representations have known an unprecedented popularity thanks to *diffusion models*. The idea behind them is to transform a high-dimensional distribution (here, 3D frames) into a simpler one, and to learn how to reverse the process. This makes it possible to generate backbones by starting from random coordinates and iteratively denoising it [Watson et al., 2023]. It is of particular interest for design, as this backbone is the input of the structure-based design process. Compared with previous approaches, both the classical ones based on fragment assemblies [Huang et al., 2016; Dou et al., 2018], and the DL-ones using generative models, either GANs [Anand and Huang, 2018] or VAE [Guo et al., 2020; Eguchi et al., 2020; Lin et al., 2021], diffusion models result in more designable backbones, *i.e.*, they can be realized by a protein sequence [Lee et al., 2022]. More broadly, diffusion models have gained in popularity over other generative models as they have a better training stability, sample quality and diversity.

Even more interesting for protein design in practice is conditioned backbone generation: generating a backbone presenting the desired secondary structures or hosting the functional site of interest. The diffusion process can be constrained toward target secondary structure using blocks indicating which position should belong to a given secondary structure [Anand and Achim, 2022; Lee et al., 2022]. RFdiffusion offers even more control over the designed backbone [Watson et al., 2023]: it can also be used to design protein binder, symmetric oligomer or enzyme active site. Chroma [Ingraham et al., 2022] handles various topology specifications too, as well as text caption such as "Protein with CHAD domain" or "Crystal structure of aminotransferase".

If Chroma and RFdiffusion are coupled with a dedicated neural net for the design task (respectively a GNN and ProteinMPNN), other methods implement an inpainting strategy to make the diffusion process generates both the backbone and a corresponding sequence [Anand and Achim, 2022; Lee et al., 2022]. It should be noted that ProteinSGM [Lee et al., 2022] does not represent the structure with its coordinates but rather as an image (similar to contact maps).

Applied Successes

The use of deep learning for protein design is very recent. The majority of structure-based approaches focus on methodological advances: they present a design method applicable on any (globular) protein, and offer some evidence — *in silico* and more rarely experimentally — that the resulting sequences will indeed fold onto the target structure. However, practical design goes beyond the inverse folding problem. Still, deep learning has proven to be successful on few applied problems. With the popularity of DL-methods for design, one can only expect to see many more successes in

the near future.

Protein design targets a function rather than a structure, which is only used as a convenient proxy for function. When the goal is to improve an existing protein, its structure can be used as input of the design network. For instance, Mutcompute improved an enzyme degrading PET plastics, both in activity and thermostability [Lu et al., 2022]. It is based on a 3DCNN trained to predict the probability of an amino acid given its local environment [Shroff et al., 2020], and was used to identify stabilizing mutations that were tested experimentally. The discovery of beneficial mutations for enzyme could also be applied to antigen stabilization [Byrne and McLellan, 2022].

De novo design presents the additional difficulty of crafting the input backbone. Hallucinations of proteins structure prediction networks makes it possible to craft backbones with functional sites [Wang et al., 2022]. It was applied to the design of luciferase, an enzyme of interest for biomedical imaging [Yeh et al., 2023b]. A substrate-binding pocket and an active site were introduced into an existing scaffold by trRosetta hallucination. Then the resulting backbone was designed with either Rosetta or ProteinMPNN, and the designed sequences showed activity with a non-natural substrate.

2.3.2 Sequence-based Design

Using a structure as input is convenient, but the generation of this backbone can be a limiting step. Pure sequence-based approaches bypass this need by designing sequences solely from other sequences. In this case, training datasets can be extremely large: more than two billions sequences have been clustered in the "Big Fantastic Database" [Jumper et al., 2021] or in Mgnify [Richardson et al., 2023]. These two databases have been recently combined [Mirdita et al., 2022]. A tiny fraction of sequences within are associated with a reliable functional label (80,000 with Molecular Function Gene Ontology in SwissProt [Consortium, 2021]).

Since neural nets only accept tensors for input and output, protein sequences must be formatted into a tensor. The most straightforward way is one-hot encoding: each amino acid is represented by a Boolean vector of size 20 (or more if gaps/unknown amino acids need to be represented), all concatenated into a $n \times 20$ matrix representing the full sequence, with n the length of the sequence. This simple encoding is widely used in DL protein design methods based on generative models, including language models (LM).

Generative Models

Generative models can be used to design new sequences that should carry the function of the input training set, usually an MSA built from one chosen functional protein. This has already been proven to be feasible using generative probabilistic

models (thus using no DL) such as Markov Random Fields [Tian et al., 2018; Cheung and Yu, 2019; Russ et al., 2020] and auto-regressive models [Trinquier et al., 2021], closely related to Bayesian networks [Srinivas, 1993]. While they explicitly target a function, these methods seem intrinsically limited to sampling the distribution generated by nature, which is not ideal to design proteins offering new functions, or to extend the scope of an existing function to non-natural conditions.

All kind of generative models have been used to design sequences, including GANs [Repecka et al., 2021] to design new functional sequences (here a malate dehydrogenase MSA was used as a starting point), VAE [Greener et al., 2018] to craft metal-binding site to sequences or to predict sequences folding according to a general topological description and an encoder/decoder architecture [Wu et al., 2020] to design peptide signal sequences. These approaches have been recently reviewed [Wu et al., 2021].

These models are fed with one hot-encoding, which is a limited representation that does not integrate much information *per se*. It makes all the configurations equidistant, whereas some sequences are biologically or physically closer than others. Learning meaningful protein representations — or embedding — is a task by itself.

Learning Protein Representation

A more informative input than a one-hot vector can be obtained through a suitable sequence embedding, a fixed-size vector representing the sequence, learned to capture important information. Several approaches inspired from Natural Language Processing (NLP) tried to decipher "the language of life" [Heinzinger et al., 2019] by considering the protein sequence as a sentence and the amino acids as words. A natural sequence corresponds to a meaningful and correct sentence. In this context, "next word prediction" approaches can be directly leveraged to predict the next amino acid or to recover a masked sequence [Rives et al., 2021].

This tight connection with NLP has attracted a lot of interest and many NLP approaches have been adapted to protein. In order of publication, noteworthy approaches include protein word embeddings [Yang et al., 2018]; UniRep [Alley et al., 2019], based on recurrent architectures; SeqVec [Heinzinger et al., 2019], based on Embedding from Language Models [Peters et al., 2018]; ESM [Rives et al., 2021], based on the attention-based Transformer. Finally, ProtTrans [Elnaggar et al., 2021] compared 6 successful NLP architectures on a dataset of an unprecedented size (2,122 million proteins, 8 times larger than those previously used). They all produce a fixed-length embedding for each amino acid, then average them to obtain the sequence embedding.

One major drawback of protein embedding is the computational cost to learn them. The most complex of the models mentioned above, ProtTrans, was trained using Summit, the world's second fastest computer. Such costs are prohibitive for

most research groups, meaning the learned protein embeddings can have an impact only if the trained model is publicly available, as it is the case for ProtTrans.

If the embeddings are meaningful by themselves (the embedding space can be clustered along protein functional, biophysical and structural properties [Elnaggar et al., 2021; Rives et al., 2021]), they are especially interesting when used as inputs for subsequent supervised tasks. Indeed, simple models on top of such embedding outperformed complex models taking one-hot encoded amino acids as inputs on molecular function prediction tasks [Villegas-Morcillo et al., 2020].

With the goal of providing more meaningful protein representation for downstream predictive task, some approaches enriched the embedding with structural information. One of them built a representation only from structure by considering the multiple level of a protein (primary, secondary, tertiary) [Hermosilla Casajús et al., 2021], while another fused the sequence representation from a language model with a structural representation learned under self-supervision by predicting inter-residue distances and dihedral angles from known structures [Chen et al., 2023]. Both outperformed pure LM representations, including on predicting enzyme classification.

Design with Language Models

Beyond protein representation learning, language models can also be adapted to generate new sequences.

ProGen [Madani et al., 2020], its extension ProGen2 [Nijkamp et al., 2022] and ProtGPT2 [Ferruz et al., 2022] are autoregressive: the next amino acid is decoded based on the context of already-predicted residues. ProtGPT2 was shown to generate realistic sequences, diverse and distant from existing ones, that are predicted to fold onto ordered structure. Yet, this generation is unconstrained, resulting in limited control over the produced sequences. ProGen proposed two ways to bias the generation toward specific functions. First, conditioning it with tags describing the desired properties with keywords (such as cellular component, biological process and function terms). Second, fine-tuning the model on a set of protein of interest: in the case of lysozyme, this resulted in resulted in generated sequences with a conserved function, even rivaling that of a natural hen egg white lysozyme [Madani et al., 2021].

Computational generation can be coupled with lab experiments, either by active learning or *in silico*. Active learning aims to iteratively propose new candidates to test based on previous experimental results. A method based on GFlowNets [Jain et al., 2022] computationally assessed candidates on their performance (predicted by an oracle), diversity and novelty. It outperformed other ML methods on the generation of peptide, DNA and green fluorescent protein. UniRep embeddings have been used for *in silico* directed evolution [Biswas et al., 2021]. Embeddings were

first fine-tuned on sequences related to the target protein. Then, a low number (24 or 96) of experimentally-characterized mutants of the wild-type were used to train a linear regression predicting the activity from the fine-tuned embeddings. Finally, top sequence candidates were selected by *in silico* directed evolution based on the activity predicted by linear regression. About 10% of them displayed an activity higher than the wild-type.

2.3.3 Assessing CPD Methods

Sequences designed by any neural network (or any design methodology) need to be assessed. The soundest way to verify a sequence has the desired function or properties is through experimental validation. However, the aim of Computational Protein Design is to reduce the burden of experimental validation as much as possible. Therefore, computational evaluation is needed as well.

First, from a pure-DL perspective, computational metrics are required to select the architecture and parameters of the neural net, to define how to test its performance, and possibly to compare it with other methods. Second, once the model is decided, criteria to filter the produced sequences are needed. Finally, the selected sequences can be tested experimentally. Since it requires time and resources, this crucial step is often absent from papers, which are very recent.

This section focuses mostly on assessing structure-based methods as they are the scope of this work. Nevertheless, some of the evaluation can also be applied to sequence-based methods.

Computational Metrics

There is no perfect metric to assess computational protein design methods.

The most usual one is *Native Sequence Recovery rate (NSR)*. It evaluates whether design produces sequences similar to the native protein sequence of a given natural structure. However, with the restricted number of observed folds [Chandonia et al., 2018], it is known that many sequences will adopt a similar structure (degeneracy). Thus, a designed sequence with an NSR between 30 and 40% is already considered as satisfactory, matching the sequence identity between homologous proteins [Dawson et al., 2016].

Another common metric is *perplexity*. Once again, it is directly inspired from Natural Language Processing [Brown et al., 1992]: it measures how well a probability model predicts a sample, and it is used to evaluate language models. Perplexity can be understood intuitively on the example of protein sequence design. At each position, the model has to choose between 20 amino acids. A perplexity of 9 (as reported by the first paper to use it in the context of CPD [Ingraham et al., 2019])

means the model is as confused as if it had to pick between 9 residues. The lower the perplexity, the better (with the best possible perplexity being 1).

It is also often impossible to directly compare the NSR or the perplexity of different methods if they have not been measured (and trained, when machine learning is used) on the same sets of target backbones [Castorina et al., 2021]: some backbones are more constrained than others and this influences the likelihood of reconstructing the native sequence. The composition of the set of structures used for measuring NSR or perplexity may therefore strongly influence the final numerical estimation.

Sequence Filtering

Once the neural net is trained and validated, it can be used to produce sequences, whose quality needs to be evaluated. First, one could verify they seem realistic, for instance if they recover the natural amino acid propensity [Ferruz et al., 2022], and if they are predicted to exhibit key biological properties, such as a hydrophobic core region tightly packed with few exposed residues [Anand et al., 2022]. Finally, oracles can be used to select the sequences that are likely to present the function or property of interest [Karimi et al., 2020; Jain et al., 2022].

Another desirable property of the designed sequences is their diversity [Melnyk et al., 2022]. Indeed, experimentally testing various sequences is likely to augment the success rates. Many of the described structure-based methods are auto-regressive [Anand et al., 2022; Jing et al., 2021; Dauparas et al., 2022], so the diversity of the sequences can be controlled by the sampling temperature (defined in the paragraph on simulated annealing, page 60). However, it often results in a trade-off between quality and diversity. Other methods do not present any way of generating diversity [Gao et al., 2023], making them of little practical interest.

Finally, the recent outbreak of DL in protein structure prediction had a great impact on design. AlphaFold, the winner — with a large margin — of the CASP13 and CASP14 competitions (Critical Assessment of protein Structure Prediction) [Senior et al., 2020; Jumper et al., 2021], reaches experimental accuracy in the template modelling category (*i,e.,* when structures of homologous proteins are known). In a nutshell, AlphaFold is fed with the sequence to predict, evolutionary information in the form of a MSA, and optionally templates (structure of homologous proteins). It is based on 2 modules: EvoFormer, to reason about spatial and evolutionary relationships, and a structure module that directly outputs the 3D coordinates of structure. An iterative refinement known as recycling is used by recursively feeding the output to the same modules, with a significant impact on accuracy. In addition to the predicted structure, AlphaFold output 2 confidence scores, one local (pLDDT) and one global (pTM), as well as the predicted Aligned Error (pAE) between pairs of residues. AlphaFold was extended to deal with multimers [Evans et al., 2021], and it widely inspired following works, including the faster ESMFold [Lin et al., 2023], OmegaFold (dedicated to prediction from a single sequence) [Wu et al., 2022] and

RosettaFold [Baek et al., 2021].

The main impact of AlphaFold on design is through *forward folding*: it makes it possible to assess whether the designed sequence will fold onto the target backbone. However, there is no consensus on the metric to use: some methods use the local score (pLDDT) [Dauparas et al., 2022], other prefer the global alignment score (pTM) [Nijkamp et al., 2022] or the predicted Aligned Error (pAE). In addition, some approaches run AlphaFold with evolutionary information while others prefer the single-sequence mode.

Beyond forward folding for validation, some design architectures directly incorporate AlphaFold in the design loop using a loss based on AlphaFold output (geometry or confidence score) [Jendrusch et al., 2021; Moffat et al., 2021], but they are very costly to train.

Experimental Validation

The only way to be certain that a designed sequence displays the target structure, function or properties is to experimentally test it. This validation is quite uncommon among existing DL-based approaches because it is an expensive process (in people, time and money). Another possible explanation is these methods are very recent, and validation takes times; therefore, most papers focus on methodological development.

There are several ways to verify that the actual fold matches the target one. Circular dichroism can be used to assess whether the expected secondary structures are present [Strokach et al., 2020]. Resolving the entire structure is even more costly, and it can be done by X-ray crystallography or cryoEM or RMN spectroscopy [Anand et al., 2022] .

Adopting the target fold does not necessarily results in fulfilling the target function or properties. These have to be tested experimentally as well. For instance, thermostability is often assessed by melting temperature [Lu et al., 2022]. Assessing a function requires a specific test, which are not always easy carry out. Easy activities to assess include those resulting in a visual signal, such as green fluorescent proteins [Jain et al., 2022] and luciferase [Yeh et al., 2023b] .

Beyond the Inverse Folding Problem

Many of the described structure-based methods have a purely computational perspective, and focus only on the inverse folding problem: predicting a sequence for the input structure. It is already highly challenging from the DL point of view, as finding a suitable protein representation and an architecture to process is difficult. Yet, the goal of protein design is to obtain a function or property of interest, not to reproduce a structure.

Usually, the design target includes specific properties that are not captured by the backbone structure alone. This includes affinity, catalytic activity, ligand-specificity, ease of expression that needs specific essays. In practice, to have real added value compared to natural proteins, designed proteins often require to be functional in *non-natural* conditions in terms of temperature, pH, solvent or ligand-specificity for example. In this situation, recovering the native sequence becomes increasingly unlikely and the native sequence recovery rate (NSR), less and less relevant.

Conclusion

DL-based vs Energy-based Methods for CPD

Comparison between classical energy-based CPD methods and structure-based DL models turns out to the advantage of the latter both in terms of computational metrics [Ingraham et al., 2019; Jing et al., 2021] but of also experimental validation: ProteinMPNN was able to "save" many designs failed by Rosetta [Dauparas et al., 2022]. DL methods do not need discretized side chain geometry, and some of them explicitly model flexible backbones by using topological features instead of exact coordinates [Ingraham et al., 2019], something that is far from simple for energy-based methods [Bouchiba et al., 2021]. The ill-posed nature of the inverse folding formulation can also be avoided: design methods based on reversed structure prediction [Norn et al., 2020] seem to implicitly account for alternative backbone conformations.

However, most of the existing structure-based proposals are focused on designing sequences for a fold. In practice, various additional constraints need to be imposed, on the chemical composition (sequence), on the geometry, or the stability of various critical regions. Practical designs rarely require to produce a new sequence for a known fold. Pure sequence-based approaches starting from a known family sharing a function do target a function, but may be limited to sampling the distribution of natural proteins. Instead, computational protein design is most useful when radically new functions or properties need to be created.

Therefore, one of the main weaknesses of DL approaches to protein design probably lies in the difficulty they have to produce specifically targeted out-of-distribution protein sequences which would fold and work in non-natural conditions in terms of pH, temperature, ligand-specificity, target, catalytic activity or other enhanced properties. This is often dealt with, in energy-based design, using multiple criteria or constraints capturing not only energy, but also design targets. Some DL approaches have tried to incorporate constraints to their model by adding conditioning tags [Madani et al., 2020] or additional features, such as a supplementary one-hot vector indicating the desired type of metallo-binding site to add [Greener et al., 2018]. Imposing constraints on DL models output is a challenging problem for Deep Learning in general, for which specific but expensive losses have been proposed [Xu et al., 2018]. This usually requires a new training every time a new constraint needs to be enforced.

Objective of the Thesis

The main goal of this thesis is to combine the specific strengths of both the DL-based and the energy-based approaches for computational protein design.

More specifically, we aim to add a DL component to the already existing CFN-like reasoning methods. It will be in charge of decoding the sequence-structure

relationships from the mass of available data and to encode this information so that it can be used in place of existing energy functions. This implies to train a neural network to produce an output akin to a pairwise decomposable energy. The idea behind is to simultaneously benefit from the proven ability of DL to learn from existing proteins and the ability of energy-based methods to impose additional constraints often necessary for practical design.

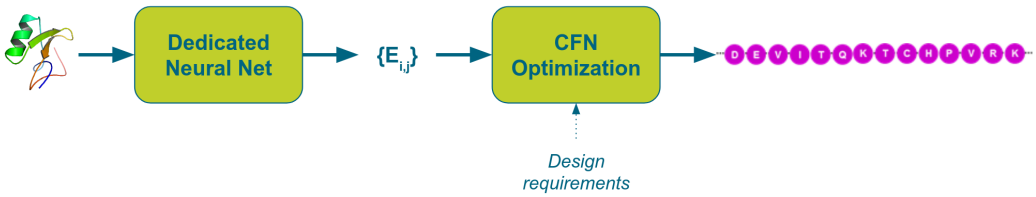


Figure 2.4: Schematic pipeline of our proposed computational protein design approach. $E_{i,j}$ is the energy scoring matrix between two arbitrary residues.

An illustration of the basic pipeline of the hybrid CPD method we proposed in this work is displayed in Figure 2.4. There are 2 main challenges to be tackled:

- **Combining automated reasoning and Deep Learning**: in a nutshell, DL relies on continuous optimization (gradient descent), while automated reasoning is based on discrete optimization. Combining both is still an open problem in artificial intelligence. This is the subject of Chapter 3.
- **Learning on protein structure data**: as detailed above, efficient learning requires both a suitable representation of data and a dedicated architecture to process them. In the case of protein structures, many representations have been proposed, and their is not a consensus on the best suited one yet. Naturally, the choice of the neural architecture is tightly linked to the data representation. Chapter 4 describes the approach we chose.

Bibliography

Adhikari, B. and Cheng, J. (2018). Confold2: improved contact-driven ab initio protein structure modeling. *BMC bioinformatics*, 19(1):1–5.

Alford, R. F., Leaver-Fay, A., Jeliazkov, J. R., O'Meara, M. J., DiMaio, F. P., Park, H., Shapovalov, M. V., Renfrew, P. D., Mulligan, V. K., Kappel, K., et al. (2017). The rosetta all-atom energy function for macromolecular modeling and design. *Journal of chemical theory and computation*, 13(6):3031–3048.

Alley, E. C., Khimulya, G., Biswas, S., AlQuraishi, M., and Church, G. M. (2019). Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322.

Allouche, D., André, I., Barbe, S., Davies, J., de Givry, S., Katsirelos, G., O'Sullivan, B., Prestwich, S., Schiex, T., and Traoré, S. (2014). Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79.

Anand, N. and Achim, T. (2022). Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*.

Anand, N., Eguchi, R., Mathews, I. I., Perez, C. P., Derry, A., Altman, R. B., and Huang, P.-S. (2022). Protein sequence design with a learned potential. *Nature communications*, 13(1):746.

Anand, N. and Huang, P. (2018). Generative modeling for protein structures. In *NeurIPS'18*.

Anishchenko, I., Chidyausiku, T. M., Ovchinnikov, S., Pellock, S. J., and Baker, D. (2020). De novo protein design by deep network hallucination. *bioRxiv*.

Baek, M., DiMaio, F., Anishchenko, I., Dauparas, J., Ovchinnikov, S., Lee, G. R., Wang, J., Cong, Q., Kinch, L. N., Schaeffer, R. D., Millán, C., Park, H., Adams, C., Glassman, C. R., DeGiovanni, A., Pereira, J. H., Rodrigues, A. V., van Dijk, A. A., Ebrecht, A. C., Opperman, D. J., Sagmeister, T., Buhlheller, C., Pavkov-Keller, T., Rathinaswamy, M. K., Dalwadi, U., Yip, C. K., Burke, J. E., Garcia, K. C., Grishin, N. V., Adams, P. D., Read, R. J., and Baker, D. (2021). Accurate prediction of protein structures and interactions using a 3-track network. *bioRxiv*.

Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. (2000). The protein data bank. *Nucleic Acids Res.*, 28:235–242.

Biswas, S., Khimulya, G., Alley, E., and et al. (2021). Low-n protein engineering with data-efficient deep learning. *Nat Methods*, 18:389–396.

Bolon, D. N., Marcus, J. S., Ross, S. A., and Mayo, S. L. (2003). Prudent modeling of core polar residues in computational protein design. *Journal of molecular biology*, 329(3):611–622.

Bouchiba, Y., Cortés, J., Schiex, T., and Barbe, S. (2021). Molecular flexibility in computational protein design: an algorithmic perspective. *Protein Engineering, Design and Selection*, 34. gzab011.

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., Lai, J. C., and Mercer, R. L. (1992). An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–40.

Buchete, N.-V., Straub, J. E., and Thirumalai, D. (2004). Orientational potentials extracted from protein structures improve native fold recognition. *Protein Science*, 13(4):862–874.

Byrne, P. O. and McLellan, J. S. (2022). Principles and practical applications of structure-based vaccine design. *Current Opinion in Immunology*, 77:102209.

Castorina, L. V., Petrenas, R., Subr, K., and Wood, C. W. (2021). Pdbench: Evaluating computational methods for protein sequence design. *arXiv preprint arXiv:2109.07925*.

Chandonia, J.-M., Fox, N. K., and Brenner, S. E. (2018). Scope: classification of large macromolecular structures in the structural classification of proteins—extended database. *Nucleic Acids Research*, 47(D1):D475–D481.

Chen, C., Zhou, J., Wang, F., Liu, X., and Dou, D. (2023). Structure-aware protein self-supervised learning. *Bioinformatics*, 39(4):btad189.

Chen, S., Sun, Z., Lin, L., Liu, Z., Liu, X., Chong, Y., Lu, Y., Zhao, H., and Yang, Y. (2020). To improve protein sequence profile prediction through image captioning on pairwise residue distance map. *Journal of Chemical Information and Modeling*, 60(1):391–399.

Cheung, N. J. and Yu, W. (2019). Sibe: a computation tool to apply protein sequence statistics to predict folding and design in silico. *BMC bioinformatics*, 20(1):1–11.

Clark, L. A., Boriack-Sjodin, P. A., Eldredge, J., Fitch, C., Friedman, B., Hanf, K. J., Jarpe, M., Liparoto, S. F., Li, Y., Lugovskoy, A., et al. (2006). Affinity enhancement of an in vivo matured therapeutic antibody using structure-based computational design. *Protein science*, 15(5):949–960.

Consortium, T. U. (2021). Uniprot: the universal protein knowledgebase in 2021. *Nucleic Acids Res.*, 49:D1:235–242.

Cretin, G., Galochkina, T., de Brevern, A. G., and Gelly, J.-C. (2021). Pythia: Deep learning approach for local protein conformation prediction. *International Journal of Molecular Sciences*, 22(16):8831.

Dahiyat, B. I. and Mayo, S. L. (1996). Protein design automation. *Protein science*, 5(5):895–903.

Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I., Courbet, A., de Haas, R. J., Bethel, N., et al. (2022). Robust deep learning–based protein sequence design using proteinMPNN. *Science*, 378(6615):49–56.

Dawson, N. L., Lewis, T. E., Das, S., Lees, J. G., Lee, D., Ashford, P., Orengo, C. A., and Sillitoe, I. (2016). CATH: an expanded resource to predict protein function through structure and sequence. *Nucleic Acids Research*, 45(D1):D289–D295.

Defresne, M., Barbe, S., and Schiex, T. (2021). Protein design with deep learning. *International Journal of Molecular Sciences*, 22(21):11741.

Desmet, J., Maeyer, M. D., Hazes, B., and Lasters, I. (1992). The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, 356(6369):539–542.

Dou, J., Vorobieva, A. A., Sheffler, W., Doyle, L. A., Park, H., Bick, M. J., Mao, B., Foight, G. W., Lee, M. Y., Gagnon, L. A., et al. (2018). De novo design of a fluorescence-activating β-barrel. *Nature*, 561(7724):485–491.

Du, Y., Meier, J., Ma, J., Fergus, R., and Rives, A. (2020a). Energy-based models for atomic-resolution protein conformations. *arXiv preprint arXiv:2004.13167*.

Du, Y., Meier, J., Ma, J., Fergus, R., and Rives, A. (2020b). Energy-based models for atomic-resolution protein conformations.

Dunbrack Jr, R. L. (2002). Rotamer libraries in the 21st century. *Current opinion in structural biology*, 12(4):431–440.

Eguchi, R. R., Anand, N., Choe, C. A., and Huang, P.-S. (2020). Ig-vae: generative modeling of immunoglobulin proteins by direct 3d coordinate generation. *bioRxiv*.

Eismann, S., Townshend, R. J., Thomas, N., Jagota, M., Jing, B., and Dror, R. O. (2021). Hierarchical, rotation-equivariant neural networks to select structural models of protein complexes. *Proteins: Structure, Function, and Bioinformatics*, 89(5):493–501.

Eliasof, M., Boesen, T., Haber, E., Keasar, C., and Treister, E. (2021). Mimetic neural networks: A unified framework for protein design and folding.

Elnaggar, A., Heinzinger, M., Dallago, C., Rihawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., Bhowmik, D., and Rost, B. (2021). Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing.

Evans, R., O'Neill, M., Pritzel, A., Antropova, N., Senior, A., Green, T., Žídek, A., Bates, R., Blackwell, S., Yim, J., Ronneberger, O., Bodenstein, S., Zielinski, M., Bridgland, A., Potapenko, A., Cowie, A., Tunyasuvunakool, K., Jain, R., Clancy, E., Kohli, P., Jumper, J., and Hassabis, D. (2021). Protein complex prediction with alphafold-multimer. *bioRxiv*.

Ferruz, N., Schmidt, S., and Höcker, B. (2022). Protgpt2 is a deep unsupervised language model for protein design. *Nature communications*, 13(1):4348.

Fram, B., Truebridge, I., Su, Y., Riesselman, A. J., Ingraham, J. B., Passera, A., Napier, E., Thadani, N. N., Lim, S., Roberts, K., et al. (2023). Simultaneous enhancement of multiple functional properties using evolution-informed protein design. *bioRxiv*, pages 2023–05.

Fuchs, F. B., Worrall, D. E., Fischer, V., and Welling, M. (2020). Se(3)-transformers: 3d roto-translation equivariant attention networks.

Gainza, P., Sverrisson, F., Monti, F., Rodola, E., Boscaini, D., Bronstein, M., and Correia, B. (2020). Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192.

Gao, Z., Tan, C., and Li, S. Z. (2023). Pifold: Toward effective and efficient protein inverse folding. In *The Eleventh International Conference on Learning Representations*.

Goverde, C., Wolf, B., Khakzad, H., Rosset, S., and Correia, B. E. (2023). De novo protein design by inversion of the alphafold structure prediction network. *Protein science : a publication of the Protein Society*, page e4653.

Greener, J., Moffat, L., and Jones, D. (2018). Design of metalloproteins and novel protein folds using variational autoencoders. *Sci Rep*, 8(16189).

Guo, X., Tadepalli, S., Zhao, L., and Shehu, A. (2020). Generating tertiary protein structures via an interpretative variational autoencoder. *arXiv preprint arXiv:2004.07119*.

Hallen, M. A., Martin, J. W., Ojewole, A., Jou, J. D., Lowegard, A. U., Frenkel, M. S., Gainza, P., Nisonoff, H. M., Mukund, A., Wang, S., et al. (2018). Osprey 3.0: open-source protein redesign for you, with powerful new features. *Journal of computational chemistry*, 39(30):2494–2507.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Heinzinger, M., Elnaggar, A., Wang, Y., Dallago, C., Nechaev, D., Matthes, F., and Rost, B. (2019). Modeling the language of life – deep learning protein sequences. *bioRxiv*.

Hermosilla Casajús, P., Schäfer, M., Lang, M., Fackelmann, G., Vázquez Alcocer, P. P., Kozliková, B., Krone, M., Ritschel, T., and Ropinski, T. (2021). Intrinsic-extrinsic convolution and pooling for learning on 3d protein structures. In *International Conference on Learning Representations, ICLR 2021: Vienna, Austria, May 04 2021*, pages 1–16. OpenReview. net.

Hsu, C., Nisonoff, H., Fannjiang, C., and Listgarten, J. (2021). Combining evolutionary and assay-labelled data for protein fitness prediction. *bioRxiv*, pages 2021–03.

Hsu, C., Verkuil, R., Liu, J., Lin, Z., Hie, B., Sercu, T., Lerer, A., and Rives, A. (2022). Learning inverse folding from millions of predicted structures. In *International Conference on Machine Learning*, pages 8946–8970. PMLR.

Huang, B., Fan, T., Wang, K., Zhang, H., Yu, C., Nie, S., Qi, Y., Zheng, W.-M., Han, J., Fan, Z., et al. (2023). Accurate and efficient protein sequence design through learning concise local environment of residues. *Bioinformatics*, 39(3):btad122.

Huang, B., Xu, Y., Hu, X., Liu, Y., Liao, S., Zhang, J., Huang, C., Hong, J., Chen, Q., and Liu, H. (2022). A backbone-centred energy function of neural networks for protein design. *Nature*, 602(7897):523–528.

Huang, P.-S., Feldmeier, K., Parmeggiani, F., Fernandez Velasco, D. A., Höcker, B., and Baker, D. (2016). De novo design of a four-fold symmetric tim-barrel protein with atomic-level accuracy. *Nature chemical biology*, 12(1):29–34.

Ingraham, J., Baranov, M., Costello, Z., Frappier, V., Ismail, A., Tie, S., Wang, W., Xue, V., Obermeyer, F., Beam, A., et al. (2022). Illuminating protein space with a programmable generative model. *bioRxiv*, pages 2022–12.

Ingraham, J., Garg, V. K., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Jain, M., Bengio, E., Hernandez-Garcia, A., Rector-Brooks, J., Dossou, B. F., Ekbote, C. A., Fu, J., Zhang, T., Kilgour, M., Zhang, D., et al. (2022). Biological sequence design with gflownets. In *International Conference on Machine Learning*, pages 9786–9801. PMLR.

Jendrusch, M., Korbel, J. O., and Sadiq, S. K. (2021). Alphadesign: A de novo protein design framework based on alphafold. *BioRxiv*, pages 2021–10.

Jiang, L., Althoff, E. A., Clemente, F. R., Doyle, L., Rothlisberger, D., Zanghellini, A., Gallaher, J. L., Betker, J. L., Tanaka, F., Barbas III, C. F., et al. (2008). De novo computational design of retro-aldol enzymes. *science*, 319(5868):1387–1391.

Jing, B., Eismann, S., Suriana, P., Townshend, R. J. L., and Dror, R. (2021). Learning from protein structure with geometric vector perceptrons. In *International Conference on Learning Representations*.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Karimi, M., Zhu, S., Cao, Y., and Shen, Y. (2020). De novo protein design for novel folds using guided conditional wasserstein generative adversarial networks. *Journal of Chemical Information and Modeling*, 60(12):5667–5681.

Krapp, L. F., Abriata, L. A., Cortés Rodriguez, F., and Dal Peraro, M. (2023). Pesto: parameter-free geometric deep learning for accurate prediction of protein binding interfaces. *Nature Communications*, 14(1):2175.

Kuhlman, B., Dantas, G., Ireton, G. C., Varani, G., Stoddard, B. L., and Baker, D. (2003). Design of a novel globular protein fold with atomic-level accuracy. *science*, 302(5649):1364–1368.

Leaver-Fay, A., Tyka, M., Lewis, S. M., Lange, O. F., Thompson, J., Jacak, R., Kaufman, K. W., Renfrew, P. D., Smith, C. A., Sheffler, W., et al. (2011). Rosetta3: an object-oriented software suite for the simulation and design of macromolecules. In *Methods in enzymology*, volume 487, pages 545–574. Elsevier.

Lee, J. S., Kim, J., and Kim, P. M. (2022). Proteinsgm: Score-based generative modeling for de novo protein design. *bioRxiv*, pages 2022–07.

Li, Z., Yang, Y., Faraggi, E., Zhan, J., and Zhou, Y. (2014). Direct prediction of profiles of sequences compatible with a protein structure by neural networks with fragment-based local and energy-based nonlocal profiles. *Proteins: Structure, Function, and Bioinformatics*, 82(10):2565–2573.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., et al. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130.

Lin, Z., Sercu, T., LeCun, Y., and Rives, A. (2021). Deep generative models create new and diverse protein structures. In *Machine Learning for Structural Biology Workshop, NeurIPS*.

Liu, Y., Zhang, L., Wang, W., Zhu, M., Wang, C., Li, F., Zhang, J., Li, H., Chen, Q., and Liu, H. (2022). Rotamer-free protein sequence design based on deep learning and self-consistency. *Nature Computational Science*, 2(7):451–462.

López-Blanco, J. R. and Chacón, P. (2019). Korp: knowledge-based 6d potential for fast protein and loop modeling. *Bioinformatics*, 35(17):3013–3019.

Lu, H., Diaz, D. J., Czarnecki, N. J., Zhu, C., Kim, W., Shroff, R., Acosta, D. J., Alexander, B. R., Cole, H. O., Zhang, Y., et al. (2022). Machine learning-aided engineering of hydrolases for pet depolymerization. *Nature*, 604(7907):662–667.

Lu, H. and Skolnick, J. (2001). A distance-dependent atomic knowledge-based potential for improved protein structure selection. *Proteins: Structure, Function, and Bioinformatics*, 44(3):223–232.

Madani, A., Krause, B., Greene, E. R., Subramanian, S., Mohr, B. P., Holton, J. M., Olmos Jr, J. L., Xiong, C., Sun, Z. Z., Socher, R., et al. (2021). Deep neural language modeling enables functional protein generation across families. *bioRxiv*, pages 2021–07.

Madani, A., McCann, B., Naik, N., Keskar, N. S., Anand, N., Eguchi, R. R., Huang, P.-S., and Socher, R. (2020). Progen: Language modeling for protein generation.

Maier, J. A., Martinez, C., Kasavajhala, K., Wickstrom, L., Hauser, K. E., and Simmerling, C. (2015). ff14sb: improving the accuracy of protein side chain and backbone parameters from ff99sb. *Journal of chemical theory and computation*, 11(8):3696–3713.

Mallinson, S. J., Dessaux, D., Barbe, S., and Bomble, Y. J. (2023). Computer-aided engineering of a non-phosphorylating glyceraldehyde-3-phosphate dehydrogenase to enable cell-free biocatalysis. *ACS Catalysis*, 13:11781–11797.

Melnyk, I., Lozano, A., Das, P., and Chenthamarakshan, V. (2022). Alphafold distillation for improved inverse protein folding. *arXiv preprint arXiv:2210.03488*.

Mirdita, M., Schütze, K., Moriwaki, Y., Heo, L., Ovchinnikov, S., and Steinegger, M. (2022). Colabfold: making protein folding accessible to all. *Nature methods*, 19(6):679–682.

Moffat, L., Greener, J. G., and Jones, D. T. (2021). Using alphafold for rapid and accurate fixed backbone protein design. *Biorxiv*, pages 2021–08.

Ng, A. H., Nguyen, T. H., Gómez-Schiavon, M., Dods, G., Langan, R. A., Boyken, S. E., Samson, J. A., Waldburger, L. M., Dueber, J. E., Baker, D., et al. (2019). Modular and tunable biological feedback control using a de novo protein switch. *Nature*, 572(7768):265–269.

Nijkamp, E., Ruffolo, J., Weinstein, E. N., Naik, N., and Madani, A. (2022). Progen2: exploring the boundaries of protein language models. *arXiv preprint arXiv:2206.13517*.

Noguchi, H., Addy, C., Simoncini, D., Wouters, S., Mylemans, B., Van Meervelt, L., Schiex, T., Zhang, K. Y., Tame, J. R., and Voet, A. R. (2019). Computational design of symmetrical eight-bladed β-propeller proteins. *IUCrJ*, 6(1):46–55.

Norn, C., Wicky, B. I. M., Juergens, D., Liu, S., Kim, D., Koepnick, B., Anishchenko, I., Players, F., Baker, D., and Ovchinnikov, S. (2020). Protein sequence design by explicit energy landscape optimization. *bioRxiv*.

O'Connell, J., Li, Z., Hanson, J., Heffernan, R., Lyons, J., Paliwal, K., Dehzangi, A., Yang, Y., and Zhou, Y. (2018). Spin2: Predicting sequence profiles from protein structures using deep neural networks. *Proteins: Structure, Function, and Bioinformatics*, 86(6):629–633.

Packer, M. S. and Liu, D. R. (2015). Methods for the directed evolution of proteins. *Nature Reviews Genetics*, 16(7):379–394.

Park, B. and Levitt, M. (1996). Energy functions that discriminate x-ray and near-native folds from well-constructed decoys. *Journal of molecular biology*, 258(2):367–392.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations.

Pierce, N. A. and Winfree, E. (2002). Protein design is np-hard. *Protein engineering*, 15(10):779–782.

Ponder, J. W. and Case, D. A. (2003). Force fields for protein simulations. *Advances in protein chemistry*, 66:27–85.

Poole, A. M. and Ranganathan, R. (2006). Knowledge-based potentials in protein design. *Current opinion in structural biology*, 16(4):508–513.

Qi, Y. and Zhang, J. Z. H. (2020). Densecpd: Improving the accuracy of neural-network-based computational protein sequence design with densenet. *Journal of Chemical Information and Modeling*, 60(3):1245–1252.

Quijano-Rubio, A., Yeh, H.-W., Park, J., Lee, H., Langan, R. A., Boyken, S. E., Lajoie, M. J., Cao, L., Chow, C. M., Miranda, M. C., et al. (2021). De novo design of modular and tunable protein biosensors. *Nature*, 591(7850):482–487.

Repecka, D., Jauniskis, V., and Karpus, L. e. a. (2021). Expanding functional protein sequence spaces using generative adversarial networks. *Nat Mach Intell*, 3(8):324–333.

Richardson, L., Allen, B., Baldi, G., Beracochea, M., Bileschi, M. L., Burdett, T., Burgin, J., Caballero-Pérez, J., Cochrane, G., Colwell, L. J., et al. (2023). Mgnify: the microbiome sequence data analysis resource in 2023. *Nucleic Acids Research*, 51(D1):D753–D759.

Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., and Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15).

Romero, P. A. and Arnold, F. H. (2009). Exploring protein fitness landscapes by directed evolution. *Nature reviews Molecular cell biology*, 10(12):866–876.

Roney, J. P. and Ovchinnikov, S. (2022). State-of-the-art estimation of protein model accuracy using alphafold. *Physical Review Letters*, 129(23):238101.

Röthlisberger, D., Khersonsky, O., Wollacott, A. M., Jiang, L., DeChancie, J., Betker, J., Gallaher, J. L., Althoff, E. A., Zanghellini, A., Dym, O., et al. (2008). Kemp elimination catalysts by computational enzyme design. *Nature*, 453(7192):190–195.

Ruffini, M., Vucinic, J., de Givry, S., Katsirelos, G., Barbe, S., and Schiex, T. (2019). Guaranteed diversity & quality for the weighted csp. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pages 18–25. IEEE.

Russ, W. P., Figliuzzi, M., Stocker, C., Barrat-Charlaix, P., Socolich, M., Kast, P., Hilvert, D., Monasson, R., Cocco, S., Weigt, M., and Ranganathan, R. (2020). An evolution-based model for designing chorismate mutase enzymes. *Science*, 369(6502):440–445.

Samish, I. (2017). *Computational protein design*. Springer.

Samudrala, R. and Moult, J. (1998). An all-atom distance-dependent conditional probability discriminatory function for protein structure prediction. *Journal of molecular biology*, 275(5):895–916.

Senior, A., Evans, R., Jumper, J., and et al. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 577:706–710.

Sercu, T., Verkuil, R., Meier, J., Amos, B., Lin, Z., Chen, C., Liu, J., LeCun, Y., and Rives, A. (2021). Neural potts model. *bioRxiv*, pages 2021–04.

Shlyk-Kerner, O., Samish, I., Kaftan, D., Holland, N., Maruthi Sai, P., Kless, H., and Scherz, A. (2006). Protein flexibility acclimatizes photosynthetic energy conversion to the ambient temperature. *Nature*, 442(7104):827–830.

Shroff, R., Cole, A. W., Diaz, D. J., Morrow, B. R., Donnell, I., Annapareddy, A., Gollihar, J., Ellington, A. D., and Thyer, R. (2020). Discovery of novel gain-of-function mutations guided by structure-based deep learning. *ACS synthetic biology*, 9(11):2927–2935.

Shroff, R., Cole, A. W., Morrow, B. R., Diaz, D. J., Donnell, I., Gollihar, J., Ellington, A. D., and Thyer, R. (2019). A structure-based deep learning framework for protein engineering. *bioRxiv*.

Simoncini, D., Allouche, D., De Givry, S., Delmas, C., Barbe, S., and Schiex, T. (2015). Guaranteed discrete energy optimization on large protein design problems. *Journal of chemical theory and computation*, 11(12):5980–5989.

Srinivas, S. (1993). A generalization of the noisy-or model. In *Uncertainty in artificial intelligence*, pages 208–215. Elsevier.

Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A., and Kim, P. M. (2020). Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11(4):402–411.e4.

Syrlybaeva, R. and Strauch, E.-M. (2023). Deep learning of protein sequence design of protein–protein interactions. *Bioinformatics*, 39(1):btac733.

Tanaka, S. and Scheraga, H. A. (1976). Medium-and long-range interaction parameters between amino acids for predicting three-dimensional structures of proteins. *Macromolecules*, 9(6):945–950.

Tian, P., Louis, J. M., Baber, J. L., Aniana, A., and Best, R. B. (2018). Co-evolutionary fitness landscapes for sequence design. *Angewandte Chemie International Edition*, 57(20):5674–5678.

Tischer, D., Lisanza, S., Wang, J., Dong, R., Anishchenko, I., Milles, L. F., Ovchinnikov, S., and Baker, D. (2020). Design of proteins presenting discontinuous functional sites using deep learning. *bioRxiv*.

Traoré, S., Allouche, D., André, I., De Givry, S., Katsirelos, G., Schiex, T., and Barbe, S. (2013). A new framework for computational protein design through cost function network optimization. *Bioinformatics*, 29(17):2129–2136.

Trinquier, J., Uguzzoni, G., Pagnani, A., Zamponi, F., and Weigt, M. (2021). Efficient generative modeling of protein sequences using simple autoregressive models.

Van Laarhoven, P. J., Aarts, E. H., van Laarhoven, P. J., and Aarts, E. H. (1987). *Simulated annealing*. Springer.

Vander Meersche, Y., Cretin, G., de Brevern, A. G., Gelly, J.-C., and Galochkina, T. (2021). Medusa: prediction of protein flexibility from sequence. *Journal of molecular biology*, 433(11):166882.

Vanommeslaeghe, K., Hatcher, E., Acharya, C., Kundu, S., Zhong, S., Shim, J., Darian, E., Guvench, O., Lopes, P., Vorobyov, I., et al. (2010). Charmm general force field: A force field for drug-like molecules compatible with the charmm all-atom additive biological force fields. *Journal of computational chemistry*, 31(4):671–690.

Verges, A., Cambon, E., Barbe, S., Salamone, S., Le Guen, Y., Moulis, C., Mulard, L. A., Remaud-Siméon, M., and André, I. (2015). Computer-aided engineering of a transglycosylase for the glucosylation of an unnatural disaccharide of relevance for bacterial antigen synthesis. *ACS Catalysis*, 5(2):1186–1198.

Villegas-Morcillo, A., Makrodimitris, S., van Ham, R. C. H. J., Gomez, A. M., Sanchez, V., and Reinders, M. J. T. (2020). Unsupervised protein embeddings outperform hand-crafted sequence and structure features at predicting molecular function. *Bioinformatics*, 37(2):162–170.

Vorberg, S., Seemayer, S., and Söding, J. (2018). Synthetic protein alignments by ccmgen quantify noise in residue-residue contact prediction. *PLoS computational biology*, 14(11):e1006526.

Votteler, J., Ogohara, C., Yi, S., Hsia, Y., Nattermann, U., Belnap, D. M., King, N. P., and Sundquist, W. I. (2016). Designed proteins induce the formation of nanocage-containing extracellular vesicles. *Nature*, 540(7632):292–295.

Vucinic, J., Simoncini, D., Ruffini, M., Barbe, S., and Schiex, T. (2020). Positive multistate protein design. *Bioinformatics*, 36(1):122–130.

Wang, J., Cao, H., Zhang, J., and et al. (2018). Computational protein design with deep learning neural networks. *Sci Rep*, 8(6349).

Wang, J., Lisanza, S., Juergens, D., Tischer, D., Watson, J. L., Castro, K. M., Ragotte, R., Saragovi, A., Milles, L. F., Baek, M., et al. (2022). Scaffolding protein functional sites using deep learning. *Science*, 377(6604):387–394.

Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., et al. (2023). De novo design of protein structure and function with rfdiffusion. *Nature*, pages 1–3.

Wicky, B., Milles, L., Courbet, A., Ragotte, R., Dauparas, J., Kinfu, E., Tipps, S., Kibler, R., Baek, M., DiMaio, F., et al. (2022). Hallucinating symmetric protein assemblies. *Science*, 378(6615):56–61.

Wu, R., Ding, F., Wang, R., Shen, R., Zhang, X., Luo, S., Su, C., Wu, Z., Xie, Q., Berger, B., et al. (2022). High-resolution de novo structure prediction from primary sequence. *BioRxiv*, pages 2022–07.

Wu, Z., Johnston, K. E., Arnold, F. H., and Yang, K. K. (2021). Protein sequence design with deep generative models. *Current Opinion in Chemical Biology*, 65:18–27.

Wu, Z., Yang, K. K., Liszka, M. J., Lee, A., Batzilla, A., Wernick, D., Weiner, D. P., and Arnold, F. H. (2020). Signal peptides generated by attention-based neural networks. *ACS Synthetic Biology*, 9(8):2154–2161.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Broeck, G. (2018). A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR.

Yang, J., Anishchenko, I., Park, H., Peng, Z., Ovchinnikov, S., and Baker, D. (2020). Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences*, 117(3):1496–1503.

Yang, K. K., Wu, Z., Bedbrook, C. N., and Arnold, F. H. (2018). Learned protein embeddings for machine learning. *Bioinformatics*, 34(15):2642–2648.

Yeh, A. H.-W., Norn, C., Kipnis, Y., Tischer, D., Pellock, S. J., Evans, D., Ma, P., Lee, G. R., Zhang, J. Z., Anishchenko, I., et al. (2023a). De novo design of luciferases using deep learning. *Nature*, 614(7949):774–780.

Yeh, A. H.-W., Norn, C., Kipnis, Y., Tischer, D., Pellock, S. J., Evans, D., Ma, P., Lee, G. R., Zhang, J. Z., Anishchenko, I., et al. (2023b). De novo design of luciferases using deep learning. *Nature*, 614(7949):774–780.

Zhang, Y., Chen, Y., Wang, C., Lo, C.-C., Liu, X., Wu, W., and Zhang, J. (2020). Prodconn: Protein design using a convolutional neural network. *Proteins: Structure, Function, and Bioinformatics*, 88(7):819–829.

Zhou, H. and Skolnick, J. (2011). Goap: a generalized orientation-dependent, all-atom statistical potential for protein structure prediction. *Biophysical journal*, 101(8):2043–2052.

Zhou, H. and Zhou, Y. (2002). Distance-scaled, finite ideal-gas reference state improves structure-derived potentials of mean force for structure selection and stability prediction. *Protein science*, 11(11):2714–2726.

Chapter 3

Coupling Deep Learning and Automated Reasoning

Introduction

Integrating learning and reasoning is one of the most promising path toward a general AI [Hochreiter, 2022] and therefore one of the major challenge in AI [Bengio et al., 2021]. A parallel is very often drawn with the two modes of human thoughts theorized by the psychologist Daniel Kahneman [Kahneman, 2011]: "System 1" is fast and about perception, much like Deep Learning, while "System 2" is slower and logical, like reasoning. The integration of logical reasoning with Deep Learning is the object of neuro-symbolic (NeSy) methods.

Reasoning is performing logical inference, *i.e.*, inferring new facts from known ones. Many NeSy approaches relies on first-order-like logic, including DeepProbLog [Manhaeve et al., 2018], Logic Tensor Network [Badreddine et al., 2022] or NeurASP [Yang et al., 2020]. It is out of the scope of this work, as we rely on constraint programming discrete optimization, which is within the framework of propositional logic. The notion of Cost Function Network (CFN) we work with is an extension of the SAT framework, where one tries to satisfy constraints between variables. CFNs are defined by weighted constraints (we usually call them costs) and finite-domain variables. We use them to solve weighted constraint satisfaction problems, a discrete optimization task which is the reasoning compound of our framework.

Turning a real-life situation into rules or constraints to apply propositional logic is a hard task requiring expert knowledge. The goal of NeSy methods is to automatize it using Deep Learning to extract knowledge out of the environment. The integration of DL requires sub-symbolic representations (such as vectors or tensors), obtained via the third paradigm underpinning AI: probability [Raedt et al., 2020].

Facts can be associated with a probability to be true [De Raedt and Kimmig, 2015] and constraints can be relaxed into weighted constraints that are real-valued instead of Boolean [Xu et al., 2018]. CFNs fit well within this framework as they are based on weighted constraints and they are related to Markov Random Fields (MRFs), a type of probabilistic Graphical Model (GM). We will use the MRF interpretation to learn the costs from data. Even though it is out of our scope, we could link our work within first order-logic-like frameworks through another type of GM, Markov Logic Networks.

Computational Protein Design is an example of a real-life decision problem. Indeed, the input protein structure defines a CFN via the specification of an energy function built from physics-based or statistical observations. This energy function defines the costs of the discrete problem, but it is never directly observed. However, it can be deduced from the input structure, and some solutions — namely the native sequence — are observed. In order to learn how to design a new sequence from an input structure, we chose to learn the energy function, and then solve the resulting discrete problem.

We defined the learning problem as an unsupervised task: the target energy is unknown (and we are not interested in imitating existing energy functions as we aim to learn a better one). Instead, we want the predicted energy, when minimized, to lead to protein sequences suited to the target backbone. The only training signal comes from the observation of native sequences. The problem of predicting high-quality solutions of a discrete reasoning problem from a new observation is often referred as Decision-Focused Learning (DFL) [Wilder et al., 2019]. We illustrated it on the problem we tackled in this work in Figure 3.1.

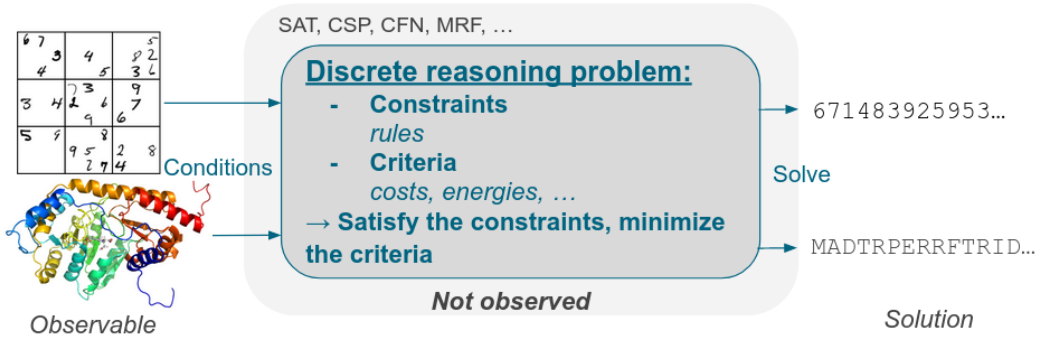


Figure 3.1: Decision-focused framework on two decision-making problems: learning to play Sudoku and learning to design proteins. In both cases, natural inputs (respectively a Sudoku grid or a protein structure) are observed together with a solution, but the parameters of the underlying problems (the rules or the energy function) are unknown. At inference, some constraints can be added *a posteriori*.

We distinguish 2 categories of DFL problems: one category has access to his-

torical data of (some of) the parameters defining the discrete problem, while the other one never observes any parameters, and tries to predict them solely from existing solutions. The first category encompasses the *Predict-then-optimize* and the *Predict-and-optimize* frameworks [Elmachtoub and Grigas, 2022; Mandi et al., 2020; Liu et al., 2023], recently reviewed in [Kotary et al., 2021]. The former regresses parameters while the latter often use regret as the loss, *i.e.,* the loss in criteria generated by using predicted instead of true values of the parameters. The CPD problem belongs instead to the second category since no ground truth energy exists.

Our formulation of learning how to design proteins requires both a hybrid learning-and-reasoning pipeline, and a neural architecture dedicated to protein data. In order to split the difficulty, we first focused on developing a pipeline to learn how to reason on simpler problems. This is the object of this chapter. We chose to focus on a Sudoku toy problem because it is a popular benchmark in the Constraint Programming community and because of its striking parallels with the CPD problem, as we will detail.

First, Section 3.1.2 introduces decision-focused learning together with the existing approaches and it describes the toy problem of Sudoku. We developed two approaches to solve this problem. The first one, presented in Section 3.2.3, directly embeds the discrete solver as a neural layer, while the second one offers a more scalable approach by decoupling training from solving. We called this approach the Emmental-PLL [Defresne et al., 2023]. It is described in Section 3.3.3 and we presented it at the International Joint Conference on Artificial Intelligence (IJCAI) 2023 in Macao, SAR.

3.1 Learning to Reason and the Sudoku Toy Problem

This section first introduces the framework of Decision-Focused Learning with no parameter history, its challenges and existing approaches. It then focuses on learning how to play Sudoku solely from example of solved grids. This problem is a popular benchmark that we also used as a toy problem to develop and test a hybrid pipeline coupling DL and discrete reasoning (DR).

3.1.1 Decision-Focused Learning

Formulation

In this work, we assume that we observe samples $(\omega, \mathbf{y})$ of the values $\mathbf{y}$ of the variables $\mathbf{Y}$ as low-cost solutions of an underlying constrained optimization problem with parameters influenced by natural inputs ω. In the case of CPD, ω will be the input protein structure and $\mathbf{y}$ a native amino-acid sequence observed on this structure (see Figure 3.2).

From a data set S of pairs $(\omega, \mathbf{y})$, we want to train a model N which predicts a discrete reasoning model $N(\omega)$ such that the observed solution y is a solution of the predicted discrete problem:

$$\mathbf{y} \in \underset{\mathbf{y} \in D^{\mathbf{Y}}}{\operatorname{argmin}} N(\omega)(\mathbf{y}) \tag{3.1}$$

This is the objective of *Decision-Focused Learning* [Wilder et al., 2019; Mandi et al., 2022]: the quality of the predictions by the model N is assessed based on the quality of the solutions resulting from the optimization of its output. In our case, the model N is a neural network with an arbitrary architecture.

The discrete reasoning model $C = N(\omega)$ defines the last layer of our hybrid neural+reasoning architecture. In our case, it is a pairwise graphical model (GM) expressed as a Cost Function Network (CFN). One good property of CFNs is their ability to express both numerical and logical functions. In order to supervise the training of the neural net N, we assume that the variables in $\mathbf{Y}$ are identified and their domains D are known. Therefore, the model N only predicts the cost functions of $N(\omega)$, that, we assume, take their value in $\bar{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$.

Additionally, we also want to exploit any information that would be available on elements of the input ω. For instance, some variables in $\mathbf{Y}$ can already be assigned, and therefore they can be directly incorporated into the discrete model $N(\omega)$. In the plain Sudoku problem, a partially filled grid of numbers is observed in ω and each observed value in the grid is known to be the value of its corresponding variable. Some information can also already be known from the natural input, or influence

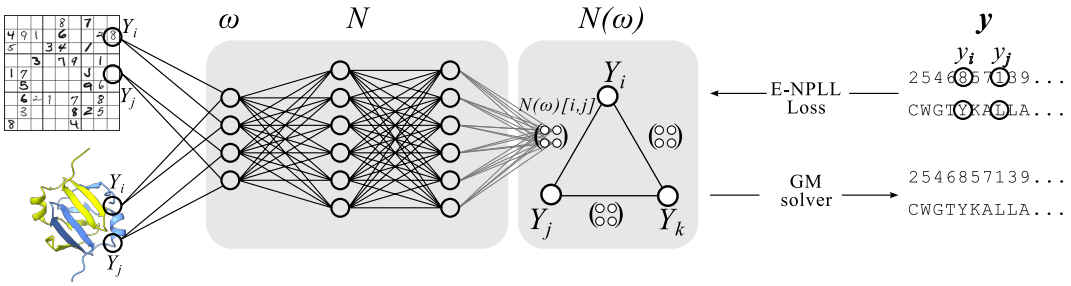


Figure 3.2: Our hybrid learning architecture: natural inputs ω (left) feed a neural net N in charge of predicting all pairwise cost functions F_{ij} of the discrete reasoning problem $N(\omega)$. Our approach is illustrated here on 2 possible problems: visual Sudoku problem and protein design (bottom).

only a subset of all the variables $\mathbf{Y}$. For instance, we will tackle a visual Sudoku task where visual inputs contributes to the rules.

Challenges

We want to train the neural net N to produce good cost functions in the sense that their joint minima is the observed solution $\mathbf{y}$. We have no historical data of these costs. Hence, the supervision is indirect and based on the objective described in equation 3.1.

To assess this objective directly, the optimal solution $\mathbf{y}^*$ of the predicted discrete model $N(\omega)$ is compared to the observed solution $\mathbf{y}$ through some loss L:

$$L(\mathbf{y}^*, \mathbf{y}) = L(\underset{\mathbf{t} \in D^{\mathbf{Y}}}{\operatorname{argmin}} N(\omega)(\mathbf{t}), \mathbf{y}) \tag{3.2}$$

The discrete solver is embedded as a neural layer, necessary to compute the training objective. The main challenge here comes from the loss L being discrete because $\mathbf{y}$ and $\mathbf{y}^*$ are discrete. Training a DL model such as our neural net N requires a *differentiable* and informative loss function. Indeed, the gradients of the loss are backpropagated to update the net's weights. A function based on the output of a discrete solver, such as $\mathbf{y}^*$, is piecewise constant [Pogančić et al., 2020]. Therefore, it has either 0 or non-existing gradients, as illustrated in Figure 3.3. In all cases, they are uninformative to train a neural network.

Therefore, the main challenge of Decision-Focused Learning is to be able to *backpropagate through the discrete solver* to train the neural network.

A second challenge comes from repeatedly calling the solver on training instances. If the optimization problem is NP-hard, it drastically limits the size of instances that the neural net can be trained on. We will see in Section 3.2.2 that tractability issues already arise on a Sudoku toy problem with 81 variables.

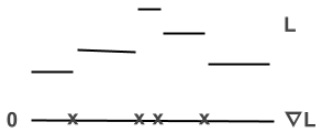


Figure 3.3: A loss function L computed based on the solution of a discrete solver is piecewise constant. Its gradients ∇L are 0 everywhere except at points of discontinuity, where they are not defined.

Existing Approaches

Existing approaches for Decision-Focused Learning with no parameter label can be divided into two groups: the first group relies on a fully-differentiable architecture, while the second group uses a dedicated loss to learn the parameters separately from the optimization.

Fully differentiable solver. Most of the proposed methods belong to the fully-differentiable class. The pioneer work OptNet [Amos and Kolter, 2017] implicitely differentiate through small quadratic programs by using the Karush-Kuhn-Tucker conditions, resulting in a convex optimization layer. More recent methods have focused on linear optimization layers. They build approximation of any discrete solver to differentiate through the piece-wise constant loss. The blackbox approach builds a continuous interpolation of the loss [Pogančić et al., 2020; Rolínek et al., 2020]. If the first version required an extra hyperparameter for trade-off between faithfulness and informativeness, as well as 2 calls to the discrete solver for each training instances, both these limitations were alleviated in a second version [Sahoo et al., 2023]. The dominant approach to turn any solver into a differentiable function is regularization, as formalized by [Blondel et al., 2020]. Methods based on perturbing the input of the solver with random noise to make it differentiable [Berthet et al., 2020; Niepert et al., 2021] are special case of regularization [Dalle et al., 2022]. Finally, for combinatorial optimization (CO), SATNet [Wang et al., 2019] offers an approach based on a convex relaxation of the discrete weighted MAXSAT problem: a semi-definite program relaxation is built and differentiated through a fixed-point condition satisfied by optimal solutions.

GNN. The pure DL approach based on Recurrent Relational Nets (RRN) [Palm et al., 2018] can also be inserted in this category. Indeed, it uses a GNN module that can be added to any graph-representation-based neural net to enable many-step relational reasoning. Each iteration of message passing represents a reasoning step; increasing the number of steps increased the neural net's reasoning abilities (up to a plateau point). A message-passing algorithm called Belief Propagation is used for inferring approximate solutions on GM [Yedidia et al., 2003], so the

decoder net of RRN can be seen as a polynomial-time solver. The main difference with Belief Propagation is that it passes embeddings of messages rather than plain message. Generally speaking, GNN and neuro-symbolic methods are tightly linked, as surveyed by [Lamb et al., 2020].

Tractability. All these methods solve the discrete problem predicted for each training instances at each epochs. On NP-hard problems, such as MaxSAT or integer linear programming, these architectures may have excruciating training costs and are therefore applied only on tiny instances (20-cities traveling salesperson problems [Pogančić et al., 2020] or 3 jobs/10 machines scheduling problems [Mandi et al., 2022]). A better tractability can be reached through convex approximations, like SATNet, but it is double-edged: approximate solving at training and inference can lead to sub-optimal performances. To keep an exact solver in the training loop, [Mulamba et al., 2021] proposed a cache of previous solution, used to approximate the optimization for some instances. Authors showed on a variety of methods that solving 5% of instances is enough to the recover same accuracy with a much lower training cost. In Section 3.2.3, we present a method based on the Hinge loss and exact solving with toulbar2. It enters this first category of approaches and indeed suffers from tractability issues.

Dedicated Loss. The second group of approaches for Decision-focused Learning relies on a dedicated loss to learn good parameters without an optimizer in the training loop. The DL-free methods by [Brouard et al., 2020] extract preferences from data. It estimated the cost functions of a CFN using an approximate log-likelihood [Park et al., 2017] and convex optimization. In Section 3.3.3 we adopt a very similar approach, but we insert DL in the pipeline, and the neural net is trained under our custom loss. One limitation of these approaches is the solver necessarily being after all the neural layers.

Multiple Solutions The majority of decision problems have several solutions. Ignoring solution multiplicity results in sub-optimal training [Nandwani et al., 2021]. However, defining a training loss in this context is non-trivial: a naive error computed on all the observed solutions would unnecessarily penalize the model. Indeed, even if it predicts a correct solution, it may differ considerably from the other observed solutions. For instance, performances of RRN [Palm et al., 2018] was shown to considerably drop on problems with multiple solutions [Nandwani et al., 2021]. Moreover, handling solution multiplicity is often coupled with dealing with incomplete information as only part of the possible solutions may be observed. In this setting, a new task was introduced: predicting any one of the possible solutions [Nandwani et al., 2021]. They proposed a generic RL formulation to enable any NeSy predictor to handle solution multiplicity.

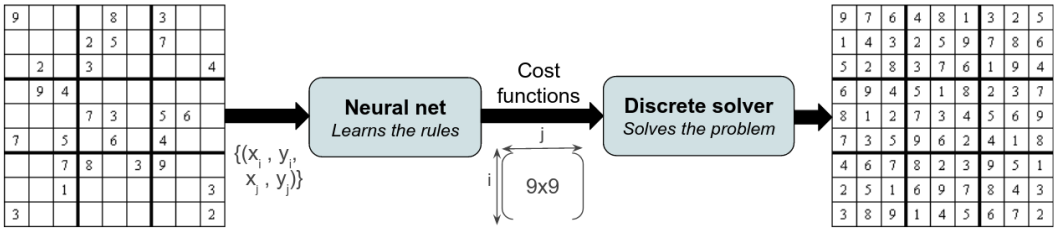


Figure 3.4: The toy problem of learning to play Sudoku. First, a neural network predicts the rule, in the form of a cost matrix, for each pair of cells. Then, a discrete solver (in our case, toulbar2) fills the grid based on the predicted grid.

3.1.2 The Sudoku Toy Problem

The NP-complete Sudoku problem is a classical logical reasoning problem that has been repeatedly used as a benchmark in a "learning to reason" context [Amos and Kolter, 2017; Wang et al., 2019; Brouard et al., 2020; Nandwani et al., 2021; Chang et al., 2020; Chen et al., 2020; Mulamba et al., 2020; Topan et al., 2021]. The task is to learn how to solve new Sudoku grids from a set of solved grids, without knowing the game rules. It is illustrated on Figure 3.4. Some approaches simplify the learning problem by using the rule structure as a prior [Palm et al., 2018] and imposing all the constraints to be the same [Franc and Savchynskyyy, 2008].

Task

Given samples $(\omega^l, \mathbf{y}^l)$ of initial and solved Sudoku grids, we want to learn how to solve new grids. Even if the rules of this game are well known (two cells on the same row, column or square cannot contain the same figure), we assume they are inaccessible during training or inference. The quality of the predicted rules is only assessed based on how well they make it possible to solve the problem.

A 9×9 Sudoku grid is represented as 81 cell coordinates with a possible hint when available. Each cell is represented by a GM variable with domain $\{1, \ldots, 9\}$. The neural net N receives the pairs of coordinates of pairs of cells (Y_i, Y_j) and predicts all pairwise cost matrices $N(\omega)[i, j]$. The neural net predicts only binary costs. There is no need to predict unary costs as an equivalent problem can be defined with binary costs only [Cooper et al., 2020]. For the plain Sudoku, hints are directly given to the solver to set the values of their corresponding variable in $N(\omega)$.

We interpret the predicted cost matrices $N(\omega)[i, j]$ as the rules of Sudoku. For pairs of cells on the same row, column or sub-square, we expect soft difference-like cost function to be predicted (a matrix with higher costs on the diagonal) that prevents the use of the same value for the two variables. Other cost matrices should be flat, indicating the absence of a pairwise constraint. For details on the representation

of Sudoku as a Cost Function Network (CFN), see the example page 39.

Sudoku players know that Sudoku grids can be more or less challenging. As one could expect, it is also harder to train how to solve hard grids than easy grids [Brouard et al., 2020; Palm et al., 2018]. We use the number of initially filled cells (hints) as a proxy to the problem hardness, a grid with fewer hints being harder. The minimal number of hints required to define a single solution is 17 [McGuire et al., 2014], defining the hardest single-solution Sudoku grids. We use an existing data set [Palm et al., 2018], composed of single-solution grids with 17 to 34 hints. We use $1,000$ grids for training and 256 for validation (both including all hardness). As in [Palm et al., 2018], we test on the hardest 17-hints instances, $1,000$ in total.

Similarities to CPD

Sudoku is of course a very simple problem, especially when compared to protein design. It is therefore more convenient to develop and test our architecture. However, we also chose Sudoku as our toy problem because it has interesting parallels with the task of designing proteins.

The Sudoku grid is a structured input, just as the protein backbone. It is composed of cells comparable to residues. The goal is to choose the identity of each cell or residue, among either 9 or 20 possibilities.

The rules governing the choice of a cell or a residue is based on pairwise interactions. Indeed, all the rules of Sudoku can be stated in terms of pairs of cells, while we write the energy function of a protein as a sum of pairwise terms. Finally, the cost functions describing those interactions depend only on coordinates. For Sudoku, rules depend on which row and column a cell is in, while physics interaction in a protein depends on atomic coordinates.

Obviously, CPD is a much more challenging problem. The resulting discrete problems are much larger, up to $10,000$ variables. This is the reason why we insist on the importance of developing a scalable training approach. In this section, the approaches we present are agnostic to the neural architecture. Therefore, as we will see in Section 4 for CPD, those methods are directly applicable to other problems that can be represented as CFNs.

Application of Existing Methods

Several of the methods for Decision-Focused Learning have been applied on the task of learning how to play Sudoku. This task is a benchmark for learning to reason.

The pioneer OptNet [Amos and Kolter, 2017] learned the rules of small 4×4 Sudoku grids. RRN [Palm et al., 2018] tackled regular 9×9 grids but they explicitly encode relationships between variables. SATNet [Wang et al., 2019] learns all the rules without any prior about their structure but they focus on easy grids with an

average of 36.2 hints. Analysis by [Brouard et al., 2020] showed that a model solving 100% of 33-hint grids can solve as few as 40% of the hardest 17-hint grids. Same authors proposed an alternative hybrid approach learning CFNs, but it is DL-free and therefore not end-to-end differentiable.

In this work, we aim for an hybrid method combining Deep Learning and CFN optimization to learn how to solve Sudoku grids of any hardness and without prior on the rules (other than them being between pairs of variables). The approaches mentioned here will be used for comparisons (except OptNet that does not scale to regular-size Sudoku grids).

Variant Tasks

Published Sudoku grids have a unique solution. Yet, if some of the initial hints are removed, several solutions will be compatible with the remaining hints. For instance, a grid with fewer than 17 hints will necessarily have several solutions. Therefore, the task of learning how to play Sudoku can be tackled in a multiple solution setting, as proposed by [Nandwani et al., 2021]. They created a dataset of Sudoku grids with multiple solutions. During training, at most 5 solutions per grid as observed, while at inference the goal is to predict any of the possible solutions.

A classical variant of the Sudoku problem is the visual Sudoku, where the hints are given as handwritten digits, such as MNIST digits [Lecun et al., 1998]. Solving visual grids is a challenging task as it requires to perform logical reasoning while dealing with uncertainties [Mulamba et al., 2020]. More challenging tasks combine learning to recognize digits with a reasoning task. DRNets learn by de-mixing overlapping visual Sudokus [Chen et al., 2020], while a very recent neuro-symbolic benchmark consist in predicting whether complete visual grids are correct or not [Augustine et al., 2022; Pryor et al., 2022]. in both cases, rules are known. Alternatively, [Brouard et al., 2020] learned the Sudoku rules on visual grids, but they used a neural net already trained to recognize digits.

The task we are interested in is to simultaneously learn how to play the game and how to recognize digit, as done by SATNet [Wang et al., 2019]. This approach was a breakthrough in integrating learning and reasoning. However, without explicit supervision, it is not able to map visual inputs (the handwritten digits) to symbolic variables (numerical digits) [Chang et al., 2020]. This mapping is known as symbol grounding. It is important to note that the combination of simultaneously learning how to play Sudoku and how to recognize digits is significantly harder than each sub-problem separately. Reusing SATNet architecture, [Topan et al., 2021] proposed a method to tackle the ungrounded visual Sudoku. The observed grids do not contain the label of the initial hints given as images. This way, the problem cannot be trivially broken into two sub-problems. Using a heavily-engineered method based on self-clustering with GANs and distillation, they reached an accuracy similar to SATNet's performance with indirect digit supervision.

3.2 Embedding the Solver as a Neural Layer

The first approach we explored to learn how to play Sudoku was to embed the solver toulbar2 as a neural layer. For each training grid, it solves the predicted Graphical Model, then the predicted solution is compared to observed solution. In order to backpropagate the discrete error into the neural net, we used the Hinge loss [Tsochantaridis et al., 2005], as described in Subsection 3.2.1. We then present the results we obtained with it and the training difficulty inherent in the method.

3.2.1 The Hinge Loss

In our problem formulation (illustrated in Figure 3.4), the parameters of the discrete problem, the Sudoku rules, are never observed. Therefore, the predicted model cannot be directly assessed. Instead, the solution $\mathbf{y}^*$ obtained by minimizing the predicted GM is compared with the observed solution $\mathbf{y}$. They can be compared by the Hamming loss.

$$L(\mathbf{y}, \mathbf{y}^*) = \mathit{Hamming}(\mathbf{y}, \mathbf{y}^*) = \sum_{i=1}^{n} \alpha \mathbb{1}[y_i \neq y_i^*]$$

where $\alpha \in \mathbb{R}_+^*$ is the weight given to each error on a variable.

The variables in $\mathbf{y}$ and $\mathbf{y}^*$ being discrete, this loss is computed on discrete points. Thus it is piece-wise constant and it cannot be directly backpropagated through the neural net. To obtain meaningful gradients, we used the Hinge loss, an informative differentiable upper bound on our target loss [Tsochantaridis et al., 2005]. The Hinge loss is :

$$\mathit{Hinge}(\omega, \mathbf{y}) = \max_{\mathbf{t} \in D^{\mathbf{Y}}} [L(\mathbf{y}, \mathbf{t}) + (N(\omega)(\mathbf{y}) - N(\omega)(\mathbf{t}))]$$

Intuitively, as solutions set further away from the observed one, their cost must increase. The Hinge loss is a contrastive loss, *i.e.,* it seeks to create a margin in costs between the true solution and others. The range of this margin is controlled by the parameter α. The Hinge loss is specifically easy to express for pairwise-decomposable losses L, such as the Hamming loss above. Indeed, it can be re-written as:

$$\mathit{Hinge}(\omega, \mathbf{y}) = N(\omega)(\mathbf{y}) - \underbrace{\min_{\mathbf{t} \in D^{\mathbf{Y}}} [N(\omega)(\mathbf{t}) - L(\mathbf{y}, \mathbf{t})]}_{\text{argmin}=\mathbf{y}^m}$$

The term $N(\omega)(\mathbf{y})$ is the cost generated by the GM predicted by the neural net. Using the decomposability of L, the term $\min_{\mathbf{t} \in D^{\mathbf{Y}}} [N(\omega)(\mathbf{t}) - L(\mathbf{y}, \mathbf{t})]$ is obtained by minimizing a perturbed problem where

the correct solution has been penalized. More precisely, we denote $F_{i,j} \in \mathbb{R}^{d \times d}$ the cost function on the pair of variables (Y_i, Y_j), *i.e.,* $F_{i,j} = N(\omega)[i, j]$. In the case of Sudoku, the domain size d is 9. Similarly, we note F_i the unary cost function on a single variable Y_i. The perturbed problem consists in adding unary costs $F_i(t_i) = -\alpha \mathbb{1}[t_i \neq y_i]$.

The perturbed problem is obtained by increasing the costs of the observed assignment of all variables Y_i to increase the gap between the costs of observed and unobserved assignments. This gap is controlled by the margin α. The perturbed problem is solved by toulbar2 (instead of the problem originally predicted by the neural net), and its optimal solution is denoted $\mathbf{y}^m$.

From the solutions of the perturbed problem, the gradients of the Hinge loss are easily computed. Indeed, the Hinge loss contains costs $N(\omega)[i, j](v_i, v_j)$ with a positive sign iff $v_i = y_i$ and $v_j = y_j$; and with a negative sign iff $v_i = y_i^m$ and $v_j = y_j^m$. The only possibly non-zero gradient terms will be therefore $+1$ for $N(\omega)[i, j](y_i, y_j)$ and -1 for $N(\omega)[i, j](y_i^m, y_j^m)$ which will cancel iff $y_i = y_i^m$ and $y_j = y_j^m$, *i.e.,* when the correct solution is found. Algorithm 1 gives the pseudo-code to compute the gradients with the Hinge loss. Those gradients are then backpropagated through the neural net to update its weights.

We note that the Hinge loss for the Hamming loss with a 0-margin (also called the contrastive Viterbi loss or the perceptron loss [LeCun et al., 2006]) is equivalent to the loss of [Sahoo et al., 2023] with no projection. Indeed, their gradient expression is the same as ours with a null margin.

Algorithm 1 Computing the gradient of the Hinge loss.

function HINGE(x, y)
  $W \leftarrow$ NeuralNet(x)
  $y^m \leftarrow$Solver(binary = W, unary = y) ▷ Solves the perturbed problem.
  Init grad to 0 ▷ Tensor of shape $(n \times n \times d \times d)$
  for pair (i, j) **do**
    $grad[i, j][y_i, y_j] += 1$
    $grad[i, j][y_i^m, y_j^m] -= 1$
  end for
  return $grad$
end function

3.2.2 Training and Tractability

Training Details

The complete training pipeline with the Hinge loss, detailed hereafter, is summarized in Figure 3.5.

The Sudoku is represented as a set of pairs of variables, each taking value in $\{1, \ldots, 9\}$. A variable is represented by its coordinates (row and column number). From this information alone, a neural network is expected to predict the correct rules for each pair of variables. It is worthy to note that weights are shared: the same neural net is applied on each pair of cells. Therefore, a single Sudoku grid is highly informative as it contains $81 * 40 = 3240$ pairs of variables.

Since the expected rules are very simple and the goal is to assess the loss, we arbitrarily fixed the architecture and we did not try to optimize it. We used a 10-layer Multi-Layer Perceptron of 128 neurons and residual connections [He et al., 2016] every 2 layers. We chose standard ReLU activations, and therefore we initialized the weights using the Kaiming initialization [He et al., 2015]. Finally, we used the Adam optimizer with a weight decay of 10^{-4} and a learning rate of 10^{-3} (other parameters take default values).

For each training grid, the predicted GM is then solved with toulbar2 using the python interface pyToulbar2 version 0.0.0.2 (with default parameters if not specified). For the solver, the costs between two variables y_i and y_j are insensitive to permutation between i and j. To incorporate this invariance within the neural net, we predict only cost matrices for pairs $(y_i, y_j)_{i<j}$ and we set $N(\omega)[j, i] = N(\omega)[i, j]^T$.

As mentioned in Section 1.3.3, CFNs are invariant to shifting and scaling of costs. Therefore, a infinite number of equivalent CFNs can be learned to correctly solve Sudoku grids. Since the optimization of a sparse GM may be more tractable, we chose to favour the CFN with 0-cost when no rule applies. We did so by applying a $L1$ regularization to the predicted GM $N(\omega)$. This regularization should not be confused with the usual regularization on the neural weight to limit over-fitting. We also applied such a regularization ($L2$, through weight decay in Adam). In addition, we quantized the representation of real costs so that close-to-zero costs are rounded.

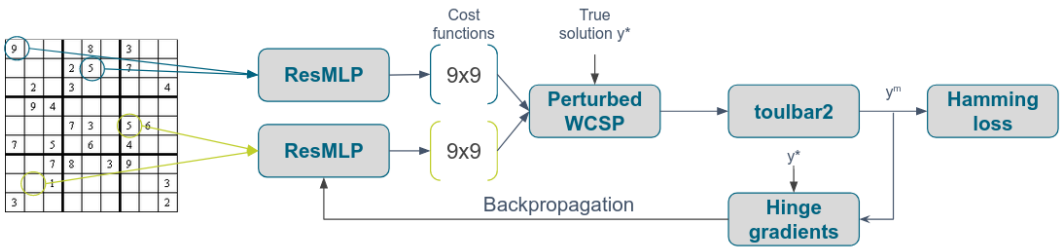


Figure 3.5: The training pipeline with the Hinge loss. For each pair of cells, a neural net predicts the corresponding cost function. The set of such function forms the GM describing the learned rules of Sudoku. Toulbar2 solves a problem with margin (true affectation are penalized with an additional cost). Then, the neural net is updated based on the Hinge loss gradients.

Intractability Issues

At the beginning of training, weights are initialized randomly. Thus, the predicted GM is random. However, random problems are notoriously difficult for exact solvers [Zhang, 2001]. Solving a single random instance can take hours and the training set contains $1,000$ grids.

The regularization term must be tuned properly: if too strong, all costs are zeros and if too weak, the regularization has no effect. We empirically observed it should be tuned together with the margin chosen in the Hinge loss. We found that a margin of 0.1 with a regularization of 5.10^{-5} worked. Due to the training difficulties with the Hinge loss, we did not explore it extensively.

Indeed, the L1-regularization is not sufficient for a tractable training since it only makes the GM sparser as training progresses. We tried and limit the solver to a partial search by limiting the number of backtracks, but it did not solve the issue. Thus, we created easier problems to train the neural net in stages of increasing difficulty. Starting from the current grid, we completed it using $\mathbf{y}$ until only k variables are to be assigned. Starting from $k = 20$, we trained the network until the percentage of solved grids from the validation set plateaued, then we increased k by 10. In the final training stage, complete problems are solved.

Tuning the discrete solver, predicting sparser GMs and increasing the difficulty of the problem throughout training makes it possible to complete full training in 2 to 3 days.

3.2.3 Results

Solving Sudokus

Approach	Acc.	#hints	Train set	Param.
[Palm et al., 2018]	96.6%	17	180,000	200k
[Wang et al., 2019]	99.8%	36.2	9,000	600k
[Brouard et al., 2020]	**100%**	17	9,000	-
Hinge (here)	**100%**	17	1,000	180k

Table 3.1: Accuracies of related works. The '# hints' gives the hardness of the test set. Param. is the number of parameters of the nets.

We compared our method with previous baselines that learned how to play Sudoku: Recurrent Relational Network (RRN) [Palm et al., 2018], SATNet [Wang et al., 2019] and a toulbar2-based approach [Brouard et al., 2020]. The comparison metric, accuracy, is the percentage of test grids fully solved. No partial reward is given for properly guessing part of a grid. As in [Palm et al., 2018], we test on

the hardest 17-hints instances, 1,000 in total. Table 3.1 gives for each approach its accuracy, the size of the training set, the difficulty of the test grids and the number of parameters in the neural net (if applicable).

We first notice the pure DL-based approach, RRN, requires much more data than hybrid approaches. Even then, it fails to solve all the hardest grids, which illustrate the difficulty of neural networks to reason. SATNet is assessed on a much easier test set composed of grids with an average of 36.2 hints instead of 17. Yet, it also fails in solving all the grids due to the approximate solving during training and inference. Both hybrid approaches using exact solving solve all the hardest grids; the advantage of the Hinge loss (and of using DL) lies in terms of data-efficiency: 9 times fewer training grids are required. We did not try to reduce the size of the training set even further.

Interpreting the Learned Model

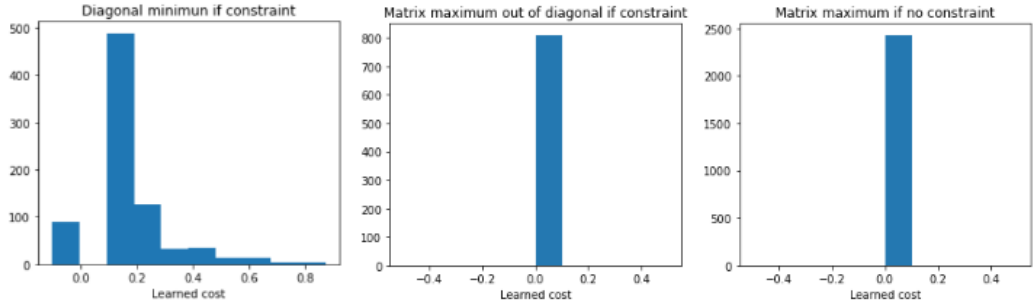


Figure 3.6: Histogram summarizing the GM learned under the Hinge loss. For pairs of variables on the same row, column or sub-square, the matrix diagonal indicates if a cost preventing them from taking the same value has been learned (left histogram), while non-diagonal costs are 0 as expected (middle histogram). Non-constrained matrices only contains 0, as expected (right histogram).

An advantage of hybrid reasoning-and-learning method is its higher interpretability, compared with a neural network alone. Indeed, the decision made can be understood by looking at the predicted GM (the solver decision is transparent as it finds the optimal one). In the case of Sudoku, we can verify the correct rules have been learned.

There are 2 categories of cost matrices to be learned: matrices representing constrained pairs of variables (that should be prevented from taking the same value), and unconstrained matrices (that should be zeros). Figure 3.6 contains 3 histograms summarizing this information. We notice that thanks to the $L1$-regularization, 0-costs are properly learned.

Interestingly, only a part of the rules are learned: some pairs on the same row, column or sub-square are not explicitly prevented by the predicted GM from the taking the same value. We empirically observed that the learned rules were sufficient for solving even the hardest Sudoku grids. This is coherent with the fact that the traditional rules of Sudoku are known to be redundant [Demoen and de la Banda, 2014]. The Hinge loss having a global view of the problem, it does not need to learn redundant constraints.

3.3 Solver-free Training with the Emmental-PLL

Despite being successful in learning how to play Sudoku, the Hinge loss is not suited to the problematic of this work due to its training time. Indeed, Sudoku is a small problem with 81 variables, while we want *in fine* to learn on proteins, whose size can go up to thousands of variables. Therefore, a more tractable method is required. Since the intractability comes from the solver being in the training loop, we explored a 2-stage approach in which a neural net is first trained apart to predict a Graphical Model, which is then optimized at inference.

In this section, we first describe how the predicted GM can be evaluated without being solved thanks to a probabilistic interpretation. We then describe the Emmental-PLL loss, a variant of the well-established pseudo log-likelihood (PLL) for MRFs [Besag, 1975] that we developed to learn in such context. Finally, we present the results obtained using the E-PLL on the Sudoku toy problem and some variants.

3.3.1 Two-stage Approach: Learning, then Optimizing

Taking the solver out of the training loop requires to be able to assess the Graphical Model (GM) predicted by the neural network without solving it. To do so, we rely on the probabilistic interpretation of a CFN as a Markov Random Field (MRF) and the estimation of MRF from data using the negative pseudo log-likelihood (NPLL), as described in Subsection 1.3.3 page 40.

From Costs to Probability

The graphical model predicted by the neural network is interpreted as an MRF in order to use the NPLL as a training loss.

$$\text{NPLL}(S) = -\sum_{y \in S} \log(\prod_i P^{\mathcal{M}}(y_i|\mathbf{y}_{-i}))$$

This requires to obtain a probability distribution $P^{N(\omega)}$ out of the costs predicted by the neural net.

For each pair of cells (Y_i, Y_j), the neural net predicts a cost function $F_{i,j} \in \mathbb{R}^{d*d}$ (in the case of Sudoku, $d = 9$). It can be defined as a cost table indicating how much an assignment of (y_i, y_j) costs. For instance, the cost of $(y_i = a, y_j = b)$ is $F_{i,j}[a, b]$.

To compute the NPLL, $P(Y_i|\mathbf{y}_{-i})$ must be calculated for each variable Y_i. This term gives the probability of each value for the cell Y_i when all the other cells are known. Since all the other variables are known, all the binary costs including Y_i can

be reduced to unary costs m_i that depend only on Y_i:

$$m_i(Y_i) = \sum_{j \neq i} F_{ij}(Y_i, y_j)$$

In a message passing interpretation, $m_i(\cdot) \in \bar{\mathbb{R}}^{|D^i|}$ represents the sum of all messages received from neighbor variables Y_j through the incident functions F_{ij}, given $Y_j = y_j$. The terms $m_i(\cdot)$ correspond to the potential functions of the MRF. For a given variable Y_i and its observed value y_i:

$$\begin{aligned} P(Y_i = y_i | y_{-i}) &= \frac{e^{-m_i(Y_i)[y_i]}}{\sum_{k=1}^{d} e^{-m_i(Y_i)[k]}} \\ &= \text{softmax}(-m_i(Y_i)) \end{aligned}$$

Therefore, each cost vector is turned into a probability using a softmax function. Computing the NPLL is in $O(n(n-1)d)$ per sample and epoch. It can easily be vectorized (computed independently on each variable).

The NPLL Fails on Logical Information

The NPLL enables a scalable training to learn a GM optimization problem from natural inputs. However, the proof of asymptotic consistency of the NPLL [Besag, 1975; Geman and Graffigne, 1986] relies on identifiability assumptions, *i.e.,* two formulations of the problem that are different lead to different sets of solutions. In the case of constraints, where the cost values are Boolean (0 for authorized, ∞ for forbidden), these assumptions do not hold because of rules redundancy. Unsurprisingly, the NPLL is known to perform poorly in the presence of large costs [Montanari and Pereira, 2009].

We observed this issue with large costs empirically. We trained the same neural net with the same amount of $L1$ regularization as in the previous section, under the supervision the NPLL. At inference, the predicted GM was solved by toulbar2. This pipeline failed at solving even the simplest Sudoku problems.

To better understand what the neural net learned, we plotted histograms summarizing the costs in Figure 3.7. We observed that only part of the constraints (left histogram) had been learned. At inference, the rules are incomplete and therefore toulbar2 fails to properly fill the grid. More precisely, the neural net systematically learns 324 constraints. Under closer inspection, it only learns rules between variables on the same row (or column depending on the weight initialization).

3.3.2 The Emmental-PLL

In order to propose a variant of the NPLL able to learn on logic information, we first needed to understand its behaviour.

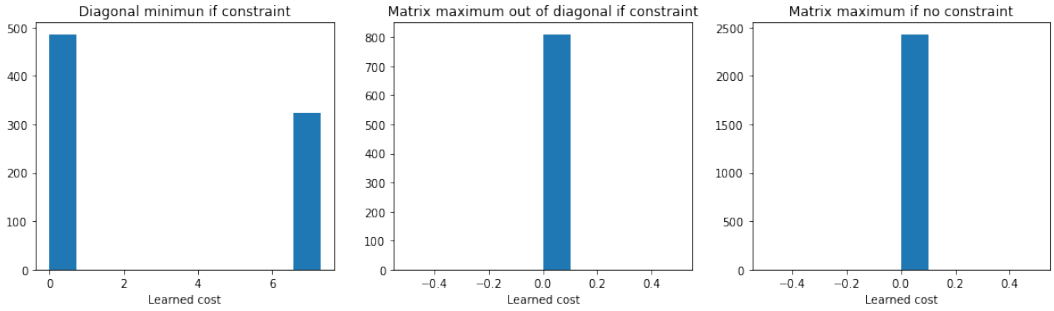


Figure 3.7: Histograms summarizing the GM learned under the NPLL supervision. The middle and right histograms show it learns 0-cost when there are no constraint (like the Hinge). However, the left histogram indicates that among all the constraints, only a subset is learned. This subset does not recapitulate of the rules, and therefore the optimization during inference fails.

Understanding the NPLL Behaviour

Intuitively, when playing Sudoku, if you have one cell to guess in the grid, looking at the row it belongs to is enough to infer its value. There is no need to additionally look at the column or sub-square.

Training with the NPLL is similar since each cell has to be guessed from the rest of the grid. Therefore, the value of the NPLL is minimal when all the rules or only the row rule have been learned. Therefore, gradients increasing costs between variables on a row or a sub-square are very small and these other rules are not learned. This intuitive explanation of the behaviour of the NPLL can be reformulated mathematically.

To understand the incapacity of the NPLL to deal with logical information, we need to look at its gradients in a context of redundant constraints. Given a pair $(\omega, \mathbf{y})$, its contribution to the NPLL gradients $\frac{\partial NPLL}{\partial N(\omega)[i,j](v_i,v_j)}$ for a given pair of values (v_i, v_j) of a pair of variables (Y_i, Y_j) is:

$$[\mathbb{1}(y_i = v_i, y_j = v_j) - P^{N(\omega)}(v_i|\mathbf{y}_{-i})\mathbb{1}(y_j = v_j)] \\ + [\mathbb{1}(y_i = v_i, y_j = v_j) - P^{N(\omega)}(v_j|\mathbf{y}_{-j})\mathbb{1}(y_i = v_i)]$$

The details of the gradient calculation can be found on Subsection 1.3.3 page 40.

The two terms in the gradient above come from NPLL terms computed on variables Y_i and Y_j respectively. Consider the example (given page 38) with four Boolean variables, $C = \{Y_1 \neq Y_2, Y_2 + Y_3 > 1, Y_3 \neq Y_4\}$ and $\mathbf{y} = (0, 1, 1, 0)$. We focus on the variables $Y_{i=2}$ and $Y_{j=3}$ and assume that C should hold under ω, which means that the pair of values $(Y_2 = 0, Y_3 = 0)$ should be predicted as forbidden. Being forbidden under ω, $\mathbb{1}(y_2 = 0, y_3 = 0) = 0$.

If additionally, the forbidden pairs $(Y_1 = 0, Y_2 = 0)$ and $(Y_3 = 0, Y_4 = 0)$ have already reached a high cost under ω, then both $P^{N(\omega)}(Y_2 = 0|\mathbf{y}_{-2})$ and $P^{N(\omega)}(Y_3 = 0|\mathbf{y}_{-3})$ will be close to zero, as well as the gradient itself. This will lead to a negligible (if any) change in the cost of the pair $(Y_2 = 0, Y_3 = 0)$: learning will be blocked or tremendously slowed down. Said otherwise, the fact that, in the context of $(\omega, \mathbf{y})$, the forbidden pair $(Y_2 = 0, Y_3 = 0)$ is redundant w.r.t. already identified forbidden pairs $(Y_1 = 0, Y_2 = 0)$ and $(Y_3 = 0, Y_4 = 0)$ effectively prevents any change in the cost $N(\omega)[2, 3](0, 0)$.

The issue with the NPLL lies therefore in the dynamic of the stochastic gradient optimization: the early identification of some high costs under ω will prevent the increase of other significant costs which are redundant in the context of the observed $\mathbf{y}$, but not redundant in the unconditioned problem.

We can now understand the behaviour of the NPLL when learning how to play Sudoku. Once the rules on the row have been learned, the rules on the columns or on the sub-squares become redundant in the context where all the cells save one are known. Therefore, under the NPLL, the other rules are not — and cannot be — learned.

Introducing the Emmental-PLL

Inspired by "dropout" in deep learning [Srivastava et al., 2014], we introduce the Emmental NPLL (E-NPLL) as an alternative to the NPLL that should still work when constraints (infeasibilities) are present in S.

The E-NPLL is a stochastic loss defined as

$$\text{E-NPLL}(\mathbf{y}) = -\sum_{Y_i \in \mathbf{Y}} \log(P^{N(\omega)}(y_i|\mathbf{y}_{-(\{i\}\cup\mathbf{M}_i)}))$$

where each $\mathbf{M}_i$ is a random subset of $\{1, \ldots, n\} \setminus \{i\}$ and $\mathbf{y}_{-(\{i\}\cup\mathbf{M}_i)}$ are values of $\mathbf{Y}_{-(\{i\}\cup\mathbf{M}_i)} = \mathbf{Y} \setminus (\mathbf{M}_i \cup \{i\})$. In this formula,$P^{N(\omega)}(y_i|\mathbf{y}_{-(\{i\}\cup\mathbf{M}_i)})$ is a short (and slighly abusive) notation for $\text{softmax}(-m_i(Y_i))$.

The idea of the E-NPLL follows directly from the previous gradient analysis: we want to prevent functions with already-learned high cost from blocking the learning of others significant and redundant (in the context of the assignment) costs by shrinking their gradients. To do so, we mask a random fraction of the predicted cost functions. When an already-learned high cost is masked, the backpropagated gradients will be able to increase the costs of the other redundant functions.

We combined this strategy with an L1 regularization on the GM predicted by the neural net to encourage the prediction of zero costs which makes the GM optimization problem easier to solve. Because the E-NPLL is designed to fight the side effects of redundant constraints on gradients, we expect it to learn a GM $N(\omega)$ with all redundant pairwise constraints.

We can now understand how the E-NPLL works. When a final cell is to be guessed, if a part of the row it belongs to is masked, one has to look at the column and/or sub-square to guess it properly. Masking some randomly-chosen cells of the grid forces the neural network to learn all the constraints in order to systematically provide a good guess.

3.3.3 Experiments

We performed various experiments to assess the supervision provided by the E-NPLL:

- Learning to play Sudoku to assess its ability to learn logical rules
- Learning in a variant setting with multiple solutions and under incomplete information.
- Learning to play Futoshiki, another grid-game with more challenging rules to learn
- Learning the visual Sudoku, where the hints are given as MNIST digits, to simulate natural inputs.

As in previous evaluations, performances are measured by the percentage of correctly filled grids; no partial credit is given for individual digits. Unless specified otherwise, all experiments used a Nvidia RTX-6000 with 24GB of VRAM and a 2.2 GHz CPU with 128 GB of RAM. We reused the same neural architecture as in Subsection 3.2.2, with no additional effort to tune it. We used the Adam optimizer with a weight decay of 10^{-4} and a learning rate of 10^{-3} (other parameters take default values). An L1 regularization with multiplier 5.10^{-4} was applied on the cost matrices $N(\omega)[i, j]$. For inference, we used toulbar2 through it python interface PyToulbar2. Code and data are available on the Forge MIA at https://forgemia.inra.fr/marianne.defresne/emmental-pll.

In all the experiments, at inference, the neural net predicts a cost matrix for each pair of cells, plus optionally unary cost vectors. The set of them form a GM that is given to the solver, together with the initial hints. The solver returns the optimal solution, which is compared to the true one. It is worth to note that the E-PLL is agnostic to both the neural architecture and the solver. They should respectively be chosen depending on the input to process and the instance format of the solver.

Sudoku

As in Subsection 3.2.1, we used a subset of the dataset from Palm et al. [2018] composed of 1000 training grids and 256 for validation, with a number of hints (representing the hardness of the grid) spanning from 17 to 34. The neural net is

trained under the sole supervision of the E-NPLL (the solver is not used during training).

As we did not tune the neural architecture, the validation set is only used to stop training. If the NPLL predicts individual cells in the context of the rest of the solution, monitoring this accuracy is not enough to ensure a sufficient set of rules has been learned. In practice, we indeed observed all the individual cells are properly predicted before the training has converged. Therefore, we chose to call the solver during validation to decide when to end the training. Fully solving the validation grids at the end of each epoch would result in the same intractability issues we faced with the Hinge loss.

Instead, our validation consists in a fast assessment of the predicted GM quality on the task of predicting a 0-cost solution of the input grid. If such solution is found, it is compared to the true one. Training is stopped when all the validation grid are solved properly under this scheme. Technically speaking, costs are quantized such that small costs are 0 and negative costs are also put to 0. Then, the solver is asked to find a 0-cost solution by setting its upper bound to 10^{-3} (our quantization level).

As previously, we then tested on the hardest 17-hints instances, $1,000$ in total. We repeated all the experiments with 10 different initialization of the weights. Average performance over those 10 training is given, along with a standard deviation. When training with the E-NPLL, messages from k randomly chosen other variables are ignored. In terms of accuracy, the training is largely insensitive to the value of the hyper-parameter k (see Table 3.2) as long as it is neither 0 (regular NPLL) nor close to $n-1$ (no information). However, larger values of k tend to lead to longer training. We set $k = 10$ for all Sudoku experiments. In this case, training takes less than 15 minutes. At inference, the predicted $N(\omega)$ leads to 100% accuracy on the $1,000$ hard test-set grids.

k	Epochs	Training time (s)	Runs with 100% test grids solved
0	100	-	0%
10	23.2 ± 2.6	566 ± 67	100%
20	38.6 ± 6.9	900 ± 151	90%
50	50.4 ± 7.6	1257 ± 184	90%
70	27.2 ± 2.7	724 ± 83	100%
80	100	-	0%

Table 3.2: Average performances over 10 initialization, for various number of E-NPLL holes k. Training is limited to 100 epochs.

In Table 3.3, we compared our results with previous approaches that learn how to solve Sudoku. The Graph Neural Net approach of Recurrent Relational Network

(RRN) [Palm et al., 2018], the convex optimization layer SATNet [Wang et al., 2019] and a hybrid ML/CP method [Brouard et al., 2020]. It should be noted that SATNet's accuracy was measured on a test set of easy Sudoku grids (avg. 36.2 hints). All the methods that rely on an exact solver were able to reach 100% accuracy but Deep Learning better exploits the inductive bias provided by the grid geometry and therefore our method has better data efficiency. In our case, with $k = 10$, we still obtained 100% accuracy on 17-hints grids using a training set with 200 grids. More epochs were necessary (less than 200) but training time did not increase ($587 \pm 32s$ over 10 training using 10 different seeds).

Approach	Acc.	#hints	Train set	Param.
Palm et al. [2018]	96.6%	17	180,000	200k
Wang et al. [2019]	99.8%	36.2	9,000	600k
Brouard et al. [2020]	**100%**	17	9,000	-
Hinge (here)	**100%**	17	1,000	180k
E-NPLL (here)	**100%**	17	**200**	180k

Table 3.3: Accuracies of related works. The '# hints' gives the hardness of the test set. Param. is the number of parameters of the nets.

As done for previous losses, we interpreted the GM learned with the E-NPLL in terms of which rules had been learned. Histograms in Figure 3.8 show that all the rules of Sudoku have been properly learned. Therefore, we can be confident that the accuracy of 100% observed on the test set actually extends to any Sudoku instance.

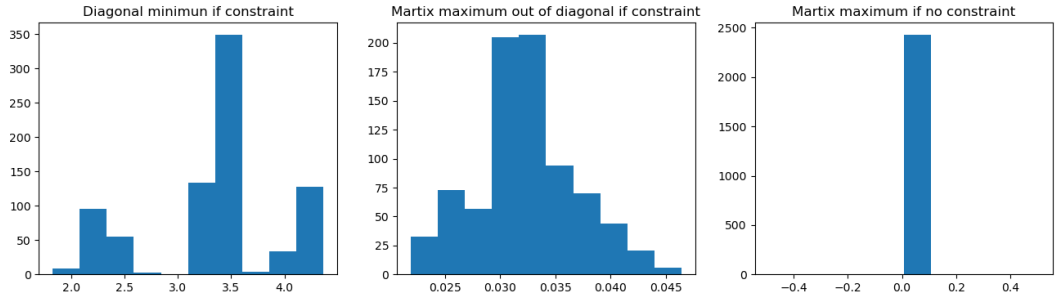


Figure 3.8: Histograms of the GM learned with the E-NPLL. As with the PLL and the Hinge loss, 0-cost are learned when there is no constraint (middle and right). Constraints between all the pairs are learned (they have strictly positive cost along the diagonal, preventing a pair of constrained variables to take the same value).

Sudoku with Multiple Solutions

In our setting, we observe instances and the associated solution $(\omega, \mathbf{y})$. However, reasoning problems usually have more than one solution. In the case where only one-

of-many solutions is observed, is our approach still able to learn a proper GM? Since the E-NPLL never compares a solver-produced solution to the provided solution $\mathbf{y}$, it should not be sensitive to the existence of many solutions.

To test this, we used the Sudoku benchmark set used in [Nandwani et al., 2021], where each grid has more than one solution. They formulated the task of Learning-One-of-Many as being able to predict any one of the solution at inference. However, the set of solutions is only partially accessible during training. For each training grid, at most 5 solutions are present in the training set. We use $1,000$; 64 and 256 grids of the data set from [Nandwani et al., 2021] respectively for training, validating, and testing. All hyper-parameters are set as previously and the validation set is only used to stop training as for plain Sudoku. The testing criteria is to be able to retrieve one of the feasible solutions (all of them are known for testing) by optimizing the predicted GM.

We used the same training procedure as with the unique-solution dataset. To compute the NPLL, we selected one of the provided solutions randomly at each epoch. As before, we repeated training for 10 different initializations of the neural net weight. On average, training took 723.4 ± 64.9 seconds (21.4 ± 1.9 epochs), leading to one of the expected solutions for 100% of the test grids. By comparison, SelectR, the RL-based method developed by [Nandwani et al., 2021], recovers one of the possible solutions for 86.7% of the test grids (see Table 3.4).

Approach	SelectR	**E-PLL**
Accuracy	86.7%	**100%**

Table 3.4: Comparison with SelectR, the approach from [Nandwani et al., 2021] on the multi-solution dataset. Accuracy refers to the ability to predict one of the possible solutions.

In fact, since the correct rules are identified with the E-NPLL, we verified that thresholding the learned costs into Boolean enables a complete enumeration of all feasible solutions for all instances in the test set.

Futoshiki

During July 2023, I co-supervised, with my supervisor T. Schiex, Romain Gambardella, an intern from an engineering school. We asked him to use the E-NPLL to try and learn the rules of Futoshiki. Like the Sudoku, the grid must be filled in such a way that there is no repetition of digit on any row, column or sub-square. In addition, some inequalities are given between pairs of cells. This rule results in cost matrices that are less sparse than with Sudoku. The diagonal still contains positive costs to prevent 2 same digits, but it also contains positive costs on the lower (resp. upper) triangular part in case of superior (resp. inferior) inequality constraint.

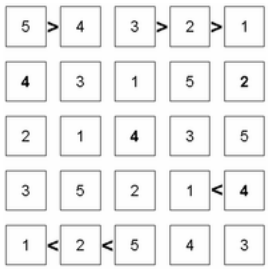


Figure 3.9: A solved Futoshiki. Bold figures are the initial hints, together with the inequalities.

We generated our own dataset, composed of 5×5 grids. Starting from an empty grid, inequalities were generated randomly. For each pair of adjacent cells, an inequality was added with a fixed uniform probability. The inequality was turned into a constraint in toulbar2, then the solver was asked to solve the grid. One grid may have several solutions. To avoid the introduction of bias by selecting one, random unary costs drawn from a uniform distribution are added to each variable. This way, a small random cost is added to each solution, helping toulbar2 to select one uniformly. This process was repeated to generate the entire dataset: 1000 training grids, 64 for validation and 200 for testing.

A grid is only represented by its inequalities, no digits are given as hints. As for Sudoku, each pair of cells is represented by its coordinates (row and column indices) and an additional feature representing the inequality (1 if $i > j$, -1 is $i < j$ and 0 if no inequality). This is fed into the same neural net as for Sudoku. Training starts with an initial learning rate of 10^{-3}, which is divided by 10 each time the NPLL computed on the validation loss increases. Training is stopped when the learning rate is below 10^{-5}.

For inference on the test grids, a threshold is applied on all the cost functions: all values below 1 are set to 0. All of the 200 grids were correctly solved.

Visual Sudoku

One strength of neural network is their ability to process natural input and handle noisy data. We simulate such a scenario by tackling the visual Sudoku problem [Franc and Savchynskyyy, 2008; Wang et al., 2019], where each hint is replaced with MNIST digits [Lecun et al., 1998], as illustrated in Figure 3.10. The goal is to simultaneously learn how to play Sudoku and how to recognize handwritten digits.

We reused our previous architecture, and we added an *untrained* LeNet network [Lecun et al., 1998] to recognize the digits. The negated logits predicted by LeNet (*i.e.,* before a softmax is applied to turn them into probabilities) were injected as unary costs into the GM. The logits were negated following the MRF definition

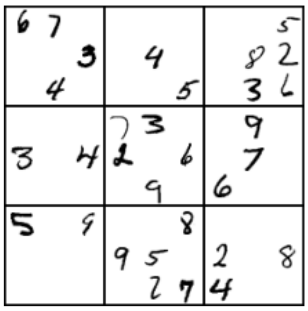


Figure 3.10: An example of a visual Sudoku input: each hint is replaced by a MNIST image.

and added to the MRF for each variable Y_i with a hint. Together with the binary costs produced by the same neural net as before, they formed the GM representing the visual Sudoku. This GM was fed to our regularized E-NPLL loss for back-propagating solutions. The complete pipeline is illustrated in Figure 3.11.

Both neural networks were trained simultaneously from random initialization with the same learning rate. No new tuning was necessary. To check for sensitivity to initialization, 10 runs with different seeds were performed. This is important as a 80% training failure rate was observed for SATNet in [Chang et al., 2020] depending on the initialization.

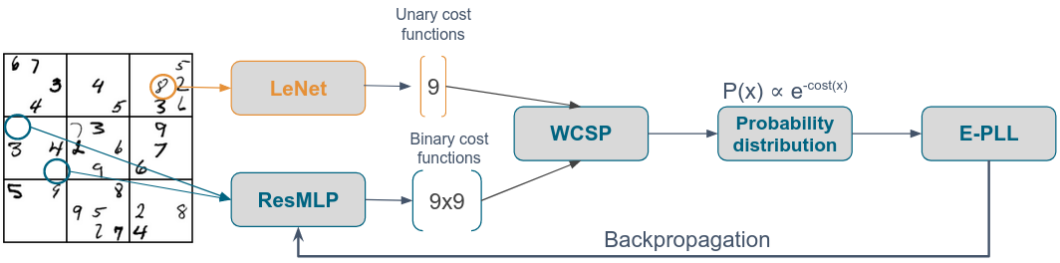


Figure 3.11: Pipeline of the visual Sudoku. A LeNet network recognizes digits and produces unary costs, while a second network predicts the rules for each pair of cells as binary costs. All these functions form a GM that is fed onto the E-NPLL.

Our data set is built from the symbolic Sudoku data set by replacing hints with corresponding MNIST images, as in [Brouard et al., 2020]. More precisely, a hash function is used to map each hint to a MNIST image so that a given grid always corresponds to the same image throughout training. Each image of a digit in the grid is known to represent the value of a single variable, observed in $\mathbf{y}$, providing grounding information to LeNet [Chang et al., 2020]. We used the same $1,000$ grids for training, the validation set contains 64 grids, only used to stop training.

After training, we extracted the LeNet network alone: it reached a 97.6% accuracy on MNIST, being indirectly supervised by the provided hints, through the E-NPLL (the supervision is direct in SATNet Chang et al. [2020]). We tested again on $1,000$ hard grids (see Table 3.5). When all the hints are correctly predicted, grids are correctly filled, as proper rules have been learned. Moreover, in 8.7% of cases, the solver was able to correct LeNet's errors, leading to an overall accuracy of 76% on hard grids. Training took an average time of $1150 \pm 13s$.

MNIST accuracy	Correctly solved grids	Of which corrected
$97.6 \pm 0.9\%$	760 ± 9	87 ± 3

Table 3.5: Visual Sudoku performance ($1,000$ hard 17-hints test grids)

We also compared our architecture with SATNet, using their data set of 9K training and 1K easy test grids (average of 36.2 hints Wang et al. [2019]). We used the exact same parameters as in the previous experiment and reused an untrained SATNet's ConvNet to process MNIST digits. We trained for at most 20 epochs (100 for SATNet) using 64 of the training grids for validation (to decide when to stop training). On this data set, SATNet's accuracy is 63.2%. SATNet's authors compared this to a theoretical maximum accuracy of 74.7%, assuming a SOTA 99.2% accuracy MNIST classifier and a perfect Sudoku solver. In our case, integrating LeNet's uncertainty on classification as negated logits in the GM pushed accuracy well beyond this theoretical limit (see Table 3.6). Since the ConvNet was trained through the E-NPLL loss, its weights are automatically adjusted to optimize the joint pseudo log-likelihood that includes also the Sudoku rules being learned. This automatically calibrates its output for the task. It is similar to what was done, *a posteriori* and with known hard Sudoku rules, in [Mulamba et al., 2020].

SATNet	Theoretical	Ours
63.2 %	74.2%	$\mathbf{94.1} \pm 0.8\%$

Table 3.6: Fraction of solved grids using SATNet data set for training and testing (averaged over 3 different initializations).

Grounding in the Visual Sudoku

The way we solve visual Sudoku so far benefits like SATNet from supervised symbol grounding [Chang et al., 2020]: they indirectly use the ground truth grid to train the MNIST classifier in a supervised fashion. Without those intermediate labels, SATNet completely fails, with a 0% accuracy. It shows an inability to map perceptions (the handwritten digits) into the corresponding symbols. In fact, SATNet

first trains the classifier thanks to the available hints in the output (a form of data leakage), then learns the Sudoku rules.

Our intern Romain Gambardella worked on training the visual Sudoku in the same setting as [Topan et al., 2021], where hints have been removed from the output to prevent them from grounding. We restrict the dataset to 17-hint grids, 1000 for training, 64 for validation and 100 for testing. Both neural nets (Le Net for digit recognition and a MLP for learning rules) are unchanged but their interface is modified: instead of using LeNet's output as unary term, it is given as additional feature in the MLP's input. The NPLL cannot be computed with missing information, so we explicitly model missing hints as unknown value. Therefore, each variable has a domain of size 10 instead of 9. As for the Futoshiki, training is stopped when the learning rate becomes too low.

This setting is significantly harder, training is unsurprisingly longer: it takes between 2 and 3 hours. Still, we reached a similar accuracy as on the ungrounded visual Sudoku, with 95 out of 100 test visual grids correctly solved. After training, LeNet has learned a permutation over digits, that is corrected by the ResMLP to predict the correct value. It is worthy to note that we place ourselves in more favourable conditions by taking only 17-digit grids (vs an average on 36.2% on SATNet's dataset). Indeed, visual grids with fewer hints are actually easier as LeNet's is less likely to make a classification mistake [Brouard et al., 2020].

In future work, we would like to avoid learning two permutations, the second correcting the first one learned by LeNet. Instead, we want to learn only the grounding permutation, that is the permutation mapping the visual perception to the digit symbols. To do so, we plan to add a one-layer perceptron that maps LeNet's output into a unary cost (that is the permutation to learn). As in [Topan et al., 2021], the ground truth value of hint is masked. Since the NPLL cannot apply on missing data, we will impute the hint with an optimization layer: toulbar2 will be given the rules learned so far to predict the hints, and these values will be backpropagated.

Conclusion

This section introduced Decision-focused learning. We developed a hybrid neural+graphical model architecture and we explored several loss functions for learning how to solve discrete reasoning problems. The whole architecture being differentiable, it makes it possible to define a graphical model from natural inputs.

First, we used the Hinge loss to incorporate the solver into the training loop. As it deals with the global problem, it is insensitive to redundant constraints and it tends to not predict them. However, repetitively solving NP-hard instance has a cost which is prohibitive when working on large instances. Therefore, this loss is not adapted to protein design.

Then, we adapted Besag's pseudo log-likelihood to enable it to learn logical information. Our Emmental-PLL removes the solver from the training loop, and therefore it is much more tractable. On various NP-hard Sudoku benchmarks, it is able to produce correct solutions from natural input (including images), while being data-efficient and capable of generalization in incomplete multiple-solution settings. During inference, an exact solver provides robust prediction. It is even able to partly correct noisy neural predictions. The E-PLL is agnostic to both the neural architecture and the solver. However, it imposes the GM prediction is the last layer of the architecture. Other approaches, including the Hinge loss or blackbox [Pogančić et al., 2020; Sahoo et al., 2023], do not suffer from this limitation. Still, it is probably a reasonable assumption for most decision-focused learning problems [Wilder et al., 2019], including protein design.

This architecture could be further explored to remove redundant constraints. We briefly tried to combine the E-PLL as pre-training and the Hinge loss to remove redundancy, but it failed as no constraint was forgotten. Moreover, as done for SATNet [Lim et al., 2022], the ultimate GM layer of our architecture could be analyzed during training to identify emerging properties such as symmetries or global decomposable constraints, allowing for more efficient learning and improved human understanding. For memory and computational efficiency, we limited ourselves to pairwise models but the use of other languages (*e.g.,* weighted clauses) in replacement of, or addition to, pairwise functions would enhance the capacity of the architecture to capture many-bodies interactions. Another possibility is the use of latent/hidden variables [Stergiou and Walsh, 1999].

We also would like to explore another way of grounding in Sudoku in order to learn the actual grounding permutation. This is the permutation mapping the visual perception to the digit symbols. To do so, we plan to add a one-layer perceptron that maps LeNet's output into a unary cost (that is the permutation to learn). As in [Topan et al., 2021], the ground truth value of hint is masked. Since the NPLL cannot apply on missing data, we will impute the hint with an optimization layer: toulbar2 will be given the rules learn so far to predict the hints, and these values

will be backpropagated.

The Hinge loss training cost could be reduced by training with an approximate solver. We experimented it with the approximate WCSP solver LR-BCD [Durante et al., 2022]. Since it is a stochastic solver, we made it predict 20 solutions for each training instance, and we chose the best one in terms of predicted cost to be backpropagated. Surprisingly, it completely fails. Closer investigation of the rules learned throughout training gives a totally different dynamic than with the exact solver, with some diagonal costs on the same matrix skyrocketing while other are zeros.

Alternative schemes where all the solutions are backpropagated, optionally weighted by their exponential negative cost, gave the same results. Worse, when starting from a trained net — thus already predicting the correct rules —, further training with LR-BCD ruined the rules learned. We ruled out the possibility of a code mistake by training with another approximate solver, toulbar2 with no allowed backtrack. The rules were not exact, but the tendency was correct (we did not spend time assessing whether they could be learned properly). The same behaviour of LR-BCD was noted on protein data, which is all the more so surprising LR-BCD gives good result in designing proteins at inference time (detailed in Chapter 5). Investigating it further could be plan for future work.

The rest of this manuscript will focus on protein design. Since our E-NPLL loss is very general, it can be directly applied to learn score functions from examples of existing protein structures. However, learning on such data sets specific challenges that will be addressed in the next section.

Bibliography

Amos, B. and Kolter, J. Z. (2017). OptNet: Differentiable optimization as a layer in neural networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR.

Augustine, E., Pryor, C., Dickens, C., Pujara, J., Wang, W., and Getoor, L. (2022). Visual sudoku puzzle classification: A suite of collective neuro-symbolic tasks. In *International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*.

Badreddine, S., Garcez, A. d., Serafini, L., and Spranger, M. (2022). Logic tensor networks. *Artificial Intelligence*, 303:103649.

Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. (2020). Learning with differentiable pertubed optimizers. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9508–9519. Curran Associates, Inc.

Besag, J. (1975). Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 24(3):179–195.

Blondel, M., Martins, A. F., and Niculae, V. (2020). Learning with fenchel-young losses. *The Journal of Machine Learning Research*, 21(1):1314–1382.

Brouard, C., de Givry, S., and Schiex, T. (2020). Pushing data into CP models using graphical model learning and solving. In *International Conference on Principles and Practice of Constraint Programming*, pages 811–827. Springer.

Chang, O., Flokas, L., Lipson, H., and Spranger, M. (2020). Assessing SATNet's ability to solve the symbol grounding problem. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1428–1439. Curran Associates, Inc.

Chen, D., Bai, Y., Zhao, W., Ament, S., Gregoire, J., and Gomes, C. (2020). Deep reasoning networks for unsupervised pattern de-mixing with constraint reasoning. In *International Conference on Machine Learning*, pages 1500–1509. PMLR.

Cooper, M., de Givry, S., and Schiex, T. (2020). Graphical models: queries, complexity, algorithms. In *Symposium on Theoretical Aspects of Computer Science, Leibniz International Proceedings in Informatics*, volume 154, pages 4–1.

Dalle, G., Baty, L., Bouvier, L., and Parmentier, A. (2022). Learning with combinatorial optimization layers: a probabilistic approach. *arXiv preprint arXiv:2207.13513*.

De Raedt, L. and Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, 100:5–47.

Defresne, M., Barbe, S., and Schiex, T. (2023). Scalable coupling of deep learning with logical reasoning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 3615–3623.

Demoen, B. and de la Banda, M. G. (2014). Redundant sudoku rules. *Theory and Practice of Logic Programming*, 14(3):363–377.

Durante, V., Katsirelos, G., and Schiex, T. (2022). Efficient low rank convex bounds for pairwise discrete Graphical Models. In *Thirty-ninth International Conference on Machine Learning*, Baltimore, United States.

Elmachtoub, A. N. and Grigas, P. (2022). Smart "predict, then optimize". *Management Science*, 68(1):9–26.

Franc, V. and Savchynskyyy, B. (2008). Discriminative learning of max-sum classifiers. *Journal of Machine Learning Research*, 9(1).

Geman, S. and Graffigne, C. (1986). Markov random field image models and their applications to computer vision. In *Proceedings of the international congress of mathematicians*, volume 1, page 2. Berkeley, CA.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hochreiter, S. (2022). Toward a broad ai. *Communications of the ACM*, 65(4):56–57.

Kahneman, D. (2011). *Thinking, fast and slow*. macmillan.

Kotary, J., Fioretto, F., Van Hentenryck, P., and Wilder, B. (2021). End-to-end constrained optimization learning: A survey. In *Proc. of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, IJCAI International Joint Conference on Artificial Intelligence, pages 4475–4482.

Lamb, L. C., Garcez, A., Gori, M., Prates, M., Avelar, P., and Vardi, M. (2020). Graph neural networks meet neural-symbolic computing: A survey and perspective. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20) Survey Track*, pages 4877–4884.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

Lim, S., Oh, E.-G., and Yang, H. (2022). Learning symmetric rules with SATNet. In *Proc. of NeurIPS'2022*.

Liu, M., Grigas, P., Liu, H., and Shen, Z.-J. M. (2023). Active learning in the predict-then-optimize framework: A margin-based approach. *arXiv preprint arXiv:2305.06584*.

Mandi, J., Bucarey, V., Tchomba, M. M. K., and Guns, T. (2022). Decision-focused learning: Through the lens of learning to rank. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 14935–14947. PMLR.

Mandi, J., Demirovic, E., Stuckey, P. J., and Guns, T. (2020). Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1603–1610.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). Deepproblog: Neural probabilistic logic programming. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

McGuire, G., Tugemann, B., and Civario, G. (2014). There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics*, 23(2):190–217.

Montanari, A. and Pereira, J. (2009). Which graphical models are difficult to learn? In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

Mulamba, M., Mandi, J., Canoy, R., and Guns, T. (2020). Hybrid classification and reasoning for image-based constraint solving. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 364–380. Springer.

Mulamba, M., Mandi, J., Diligenti, M., Lombardi, M., Lopez, V. B., and Guns, T. (2021). Contrastive losses and solution caching for predict-and-optimize. In *30th International Joint Conference on Artificial Intelligence (IJCAI-21): IJCAI-21*, pages 2833–2840. International Joint Conferences on Artificial Intelligence.

Nandwani, Y., Jindal, D., ., M., and Singla, P. (2021). Neural learning of one-of-many solutions for combinatorial problems in structured output spaces. In *International Conference on Learning Representations, ICLR'21*.

Niepert, M., Minervini, P., and Franceschi, L. (2021). Implicit MLE: Backpropagating through discrete exponential family distributions. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14567–14579. Curran Associates, Inc.

Palm, R., Paquet, U., and Winther, O. (2018). Recurrent relational networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Park, Y., Hallac, D., Boyd, S., and Leskovec, J. (2017). Learning the network structure of heterogeneous data via pairwise exponential markov random fields. In *Artificial Intelligence and Statistics*, pages 1302–1310. PMLR.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. (2020). Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*.

Pryor, C., Dickens, C., Augustine, E., Albalak, A., Wang, W., and Getoor, L. (2022). Neupsl: Neural probabilistic soft logic. *arXiv preprint arXiv:2205.14268*.

Raedt, L. d., Dumančić, S., Manhaeve, R., and Marra, G. (2020). From statistical relational to neuro-symbolic artificial intelligence. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4943–4950. International Joint Conferences on Artificial Intelligence Organization. Survey track.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. (2020). Deep graph matching via blackbox differentiation of combinatorial solvers. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M., editors, *Computer Vision – ECCV 2020*, pages 407–424, Cham. Springer International Publishing.

Sahoo, S. S., Paulus, A., Vlastelica, M., Musil, V., Kuleshov, V., and Martius, G. (2023). Backpropagation through combinatorial algorithms: Identity with projection works. In *Proc. of ICLR'23*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Stergiou, K. and Walsh, T. (1999). Encodings of non-binary constraint satisfaction problems. In *Proc. AAAI'99*, pages 163–168.

Topan, S., Rolnick, D., and Si, X. (2021). Techniques for symbol grounding with satnet. *Advances in Neural Information Processing Systems*, 34:20733–20744.

Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y., and Singer, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(9).

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. (2019). SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR.

Wilder, B., Dilkina, B., and Tambe, M. (2019). Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Broeck, G. (2018). A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR.

Yang, Z., Ishay, A., and Lee, J. (2020). Neurasp: Embracing neural networks into answer set programming. In *29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1755–1762. International Joint Conferences on Artificial Intelligence.

Yedidia, J. S., Freeman, W. T., Weiss, Y., et al. (2003). Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8(236-239):0018–9448.

Zhang, W. (2001). Phase transitions and backbones of 3-sat and maximum 3-sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 153–167. Springer.

Chapter 4

Learning Effie, an Energy Scoring Function for Design

Introduction

Computational Protein Design (CPD) aims to produce protein sequences fulfilling a function of interest for a given application. Describing the target function as an input for a computational method is a hard task, so the tight links between a protein function and its structure are exploited instead to craft a protein backbone used as a proxy input. The goal of CPD then reduces to the task of predicting sequences folding onto the input protein backbone.

We have already reported that this task starts to be well mastered by neural-based approaches. However, the real objective of CPD is a function and not a structure. Reaching it often requires to enforce some additional properties, which is still difficult to do with a neural network. Energy-based methods manage it by adding objectives and constraints to the optimization problems representing the design problem.

This work aims to bring together the advantages of both energy-based and neural-based methods by developing a new hybrid pipeline. First, a neural network, trained on existing protein structures, predicts a Graphical Model representing the interactions within the protein backbone. This GM is then optimized to obtain a protein sequence that is both stable on the target structure and meet additional optional design requirements. Thus, the produced sequences will be more likely to fulfill the target function and properties.

In this chapter, we describe the methodology to learn Effie, an Energy Function Familiarly Introduced as Effie. For a given protein backbone, Effie is a pairwise Graphical Model predicted by a neural net and that will be optimized in place of

traditional energy functions to design sequences. Effie models the probability of a sequence given the target structure. This chapter focuses on methodology; the results of applying Effie to protein design will be given in the next chapter.

As we discussed in Chapter 1, Deep Learning requires 4 main ingredients: data, a neural architecture, a loss function and an optimization process. In Chapter 3, we introduced the Emmental-NPLL, a loss to learn how to reason that we directly applied to protein design. This chapter describes the 3 other ingredients required to learn Effie.

Through the reviewing of current DL-based methods for protein design in Section 2.3.3, we noted there is no consensus so far about the most-suited representation of the input structure. The first step was therefore to choose a representation suited to the learning of a pairwise Graphical Model. This is the object of Section 4.1, that also describes the datasets we used. These data are processed by a neural architecture adapted to both leverage protein data specificity and produce a GM. The development of this architecture is related in Section 4.2. Finally, Section 4.3 focuses on the training process.

4.1 Protein Data

There is still no consensus on the best suited way to represent the input protein structure, even though graph representations are the most popular. We first list the properties we want the structure representation to have. We then describe the representation we chose, along with the features we tried. Finally, we present the two datasets we used to train and validate our model.

4.1.1 Desired Properties of the Data Representation

An adapted representation is crucial for an efficient learning [Laine et al., 2021]. Indeed, if the input data has some properties such as invariances, injecting them as inductive bias through its representation reduces the space of mapping functions learnable by the neural network. It leads to more data efficiency, notably by avoiding the need of data augmentation.

Varying-size Input

The first difficulty when working with protein data is size variation between instances. It is possible to reduce a protein to a fixed-size vector using an embedding, obtained from the sequence using a language model (see Section 2.3.2), from the structure [Hermosilla Casajús et al., 2021] or from both [Chen et al., 2023]. However, the information about the geometry would be lost.

Therefore, using a representation that allows for size variation is preferable. However, it requires a neural architecture adapted to process various input shape. It can be done via a recurrence mechanism, as done in Natural Language Processing. For instance, a MLP with sliding windows has been used to process input protein structures [Wang et al., 2018]. A graphical representation is particularly well-suited as message passing operations are designed to be insensitive to the number of neighbours [Gilmer et al., 2017].

Rotation and Translation Invariance

Protein structures are non Euclidean data. They are not as regular as an image for instance, which can be represented as a regular grid. Still, they do have some regularity properties. A global rotation or translation of the whole structure does not modify it. In particular, the interactions within the protein will be the same. The group of 3D rotations and translations is denoted SE(3). The previous property can be reformulated as SE(3)-invariance, that should be used as an inductive bias to facilitate training.

There are two ways to take into account invariances. First, the representation itself can be insensitive to the global orientation of the structure, either using

invariant features or canonized environments (see Section 2.3.3). Second, the neural architecture can use only equivariant operations, such as Tensor Field Network [Thomas et al., 2018] or SE(3)-Transformer [Fuchs et al., 2020]. It means that the output of the network net undergoes the same transformation as the input (if the output is a scalar, it will be invariant). Such architectures have been applied to small molecular graphs [Jiao et al., 2023], protein complexes [Eismann et al., 2021] and inverse folding [McPartlon et al., 2022]. Moreover, [Anand and Achim, 2022] reused the equivariant IPA attention from AlphaFold [Jumper et al., 2021] for structure generation by diffusion. Finally, some approaches mixed invariant features and equivariant message-passing operations [Jing et al., 2021].

Local Environment

The physical interactions within a protein result from local forces, such as the electrostatic interaction that decreases proportionally to the square of the distance. Therefore, to properly from a protein structure, a particular emphasis on local environments is pertinent.

For instance, all the CNN-based design methods explicitly represent the local environment of each residue, usually on a box of 20Å side centered around the target residue. Similarly, graph-based approaches define a notion of neighbouring residues, and the mechanism of message passing weakens interactions between far-apart nodes. Those examples show that local environments can be introduced both in the structure representation and in the neural architecture.

4.1.2 The Protein Structure as a Set of Residue Pairs

In our design pipeline (illustrated in Figure 2.4), the neural network takes as input a protein backbone and it outputs a pairwise Graphical Model, expressed as a Cost Function Network.

Neural Input and Output

As for Sudoku, the output GM is composed of a score matrix for each pair of residues. These matrices represent the interactions within the proteins. Since we aim for a pairwise prediction, we represent the protein as a full set of pairs of residues, similarly to what is done in the KORP potential [López-Blanco and Chacón, 2019]. This is equivalent to a fully-connected graph representation. However, the processing of the data will be quite different as we focus on pairs of residues and not on neighbourhood (see Section 4.2 for details).

The input protein backbone B is composed of the coordinates of N, C_α, C and O atoms for each of the n residues of the protein (thus, $B \in R^{n\times 4\times 3}$). To process the backbone in a way that is invariant to both translation and rotation, we chose to use invariant features. Indeed, the coordinates are not invariant. An alternative would

have been to use an equivariant architecture, but they tend to be computationally expensive [Thomas et al., 2018].

Sequence Features

We encoded information on both the sequence and the structure using features between pairs of residues. We denote B_i and B_j the backbone coordinates of two residues at the position i and j respectively, with $i, j \in \{1, \ldots, n\}$. For the sequence information, we used positional encoding of $|j - i|$ (initially introduced for Transformers Vaswani et al. [2017]) to represent the number of residues in-between the pair in the sequence.

Relative Position Features

We build invariant structural features by expressing the relative position and orientation between a pair of residues, as illustrated in Figure 4.1.

We first need to build reference frames on each residue. The α-carbon is the origin of the frame and the first axis is in the $N - C_\alpha$ direction. The other axes are built using a Gram-Schmidt ortho-normalization process applied on the $C - C_\alpha$ direction. The last axis is normal to the peptide plan and oriented such that the frame is direct.

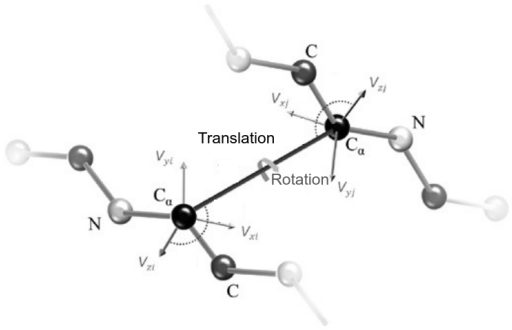


Figure 4.1: Relative position and orientation between residue B_i (bottom left) and B_j (top right). On each residue, a frame (V_x, V_y, V_z) is built. Features are built based on the translation and rotation from one frame to the other. Figure adapted from [López-Blanco and Chacón, 2019].

The relative position and orientation between two frames can be decomposed into a translation and a rotation. The translation is a direction plus a distance. For each residue, the reference point is the alpha carbon of coordinate $C\alpha$. Thus, the direction of the translation from B_i to B_j is the unit vector $\frac{C\alpha_j - C\alpha_i}{||C\alpha_j - C\alpha_i||}$ and the distance is $||C\alpha_j - C\alpha_i||$.

Expressing the rotation between B_i and B_j is trickier. We first used the same set of angles as in KORP, encoded by their sine and cosine. This resulted into $5 * 2 = 10$ features. Instead, we chose a more compact representation based on quaternion. This way to represent a 3D rotation is simpler than Euler angles and more stable that a rotation matrix [Kenwright, 2012]. A rotation consists in rotating by a certain angle around a certain axis, which corresponds to a point on a hyper sphere. This point is the quaternion $q = (w, x, y, z)$. It is also sometimes written $q = w + \vec{v}$, with $\vec{v} = (x, y, z)$ being the rotation axis and $\alpha = 2 \times arcos(w)$ the angle. A quaternion is such that $w^2+x^2+y^2+z^2 = 1$ The advantage of quaternion is that an algebra can be defined on them [Sola, 2017]. A composition of rotation then consists in multiplying quaternions, and the reverse rotation is the inverse quaternion. However, there is an ambivalence between a rotation of $+\pi$ and $-\pi$, which can perturb training [Zhou et al., 2019]. The ambivalence is contained in the sign of w: (w, x, y, z) and $(-w, x, y, z)$ represent the same rotation. Therefore, to remove ambiguity for the neural net, we systematically select the quaternion with a positive sign.

In addition to those pairwise features, we used features on each residue. Contact numbers, defined as the number of neighbours at a given threshold, are used as a proxy indicating how buried the residue is. Sine and cosine of dihedral angles are also given.

Distance Features

Variations around those relative position features were used by most of the DL-based design methods, including Structured Transformer [Ingraham et al., 2019], Geometric Vector Perceptron [Jing et al., 2021] and Protein Solver [Strokach et al., 2020].

Then in 2022, ProteinMPNN [Dauparas et al., 2022] came out. It used much simpler features, only based on distances. These features lead to superior performances, which may be explained by the fact that they avoid any discontinuity. As discussed in the next Chapter and Annex C.2, we also observed that replacing quaternions with those distances resulted in enhanced performances.

In details, these features contain distances between all the backbone atoms (N, C_α, C and O plus a virtual C_β) of the 2 residues. The virtual C_β coordinates are calculated based on the other backbone atoms coordinates using ideal angle and bond length definitions [Dauparas et al., 2022]:

$$C_\beta = -0.58273431 * a + 0.56802827 * b - 0.54067466 * c + C_\alpha$$

with $b = C_\alpha - N$, $c = C - C_\alpha$, $a = cross(b, c)$

These 25 distances are encoded with 16 Gaussian radial basis functions (GRBF) with centers evenly spaced between 0 and 20Å. For a distance d, its GRBF encoding

is:

$$\{e^{-(d-d_0)^2}\}_{d_0 \in \{0, \frac{4}{3} ..., \frac{15*4}{3}=20\}}$$

In total, there are $25 * 16 = 400$ distance features.

Handling Missing Residues

The protein structure data we use have been experimentally resolved and therefore they are exposed to missing information. If sequences are usually complete, some residues have unknown coordinates. We call them *missing residues*.

Since most of our features are based on distances, there is almost no information in the features of a pair containing (at least) one missing residue. They are particularly common on N-ter and C-ter as they usually are flexible loops. We apply a pre-processing step to cut out all missing information on the extremities of protein sequences.

Missing residues can also happen within proteins. In our first dataset (detailed hereafter), $14,000$ out of $18,000$ proteins have at least one missing residue after pre-processing, with an average of 8.3%. All the distances and angles on a pair of residues containing (at least) one missing are undefined and arbitrarily set to 0.

4.1.3 Datasets

To train our model, we use protein structures from the Protein Data Bank [Berman et al., 2000]. For each structure, at least one native sequence is known, enabling unsupervised training with the Emmental-PLL. In this work, we reused 2 existing datasets. The first one is restricted to single-chain proteins and the second one also contains protein complexes.

Single-chain Dataset

The first dataset (*single-chain*), from [Ingraham et al., 2019], is composed of single chains of proteins. Most of the DL-based design methods use this dataset for both training and testing, enabling easy comparison between methods.

In order to assess the ability of a DL model to generalize to unseen protein, the dataset should be rigorously split into training, validation and test sets such that there is no redundancy between sets. Sequence and structure similarities between proteins make this splitting non-trivial. A first filter on sequence similarity was applied such that all sequences differ from a minimal percentage (40% in this dataset). Yet, it is not enough to prevent redundancy as protein with low identity can have the same fold.

To ensure similar folds remain in the same subset, splitting was based on the CATH4.2 classification [Sillitoe et al., 2015]. All proteins with the same topology

are on the same set, which corresponds to the first 3 numbers of CAT code. For instance, all proteins with a code starting by 3.30.70 are on the same split. This resulted in a dataset of $18,024$ chains in the training set, 608 chains in the validation set, and $1,120$ chains in the test set. Proteins have at most 500 residues as longer proteins are filtered out.

The identity of an amino acid not only depends on interactions with neighbouring residues, but also with the environment. Membrane proteins, that fold into a lipidic environment, have a different composition from soluble proteins. When the solvent is not represented, the neural net may lack important information to decide which residues may fit. Using the Protein Data Bank of Transmembrane Proteins [Kozma et al., 2012], we found that the single-chain dataset contains very few transmembrane proteins (223 in the training set, 41 in the validation set and 29 in the test set). Therefore, a DL model trained on this dataset may reasonably be applied only on soluble proteins.

Finally, in order to restrict to single-chain proteins, [Ingraham et al., 2019] cut complexes into single chains. Only a part of the dataset corresponds to truly monomeric proteins. This subset, which we call *monomeric set*, contains protein shorter than 150 residues. Splitting complexes into single chains may noise the information and be misleading. Indeed, the distribution of amino acids at an interface between chains is not the same as amino acids at the surface, which are mostly hydrophilic as they are exposed to the solvant. This shortcoming is bypassed by considering multi-chain proteins.

Multi-chain Dataset

The second dataset (*multi-chain*) was developed to train ProteinMPNN [Dauparas et al., 2022]. It is composed of protein assemblies in the PDB (as of Aug 02, 2021) determined by X-ray crystallography or cryoEM with a resolution better than 3.5Å. Proteins were clustered with a 30% identity cut-off into $25,361$ clusters, which were split randomly into training (23,358), validation (1,464), and testing (1,539) sets such that chains from a given assembly are all in the same set. For each training epoch, one sequence member of each cluster is selected at random. If it corresponds to several conformations, one is randomly chosen. Then the biological assembly is reconstructed from the PDB entry.

Additional features are available on this dataset: for each residue, the chain it belongs to is known. Moreover, proteins are annotated as being soluble or in a membrane, which enables to train models specialized in soluble-only. In total, this dataset contains 3498 membrane proteins.

Training on this dataset set new challenges, as protein complexes can have up to $10,000$ residues, which raises new issues in terms of memory and computational tractability. This will be discussed in Section 4.3.

4.2 Architecture

Our neural architecture is an auto-encoder that maps an input protein backbone into a latent representation, the pairwise Graphical Model expressed as a Cost Function Network (CFN). At inference, the predicted GM is decoded by the solver toulbar2 through optimization into designed sequences. More precisely, the neural net is fed with each pair of residues in the protein, and it produces a 20×20 cost matrix for each of them.

The complete architecture is organized around two neural nets, as illustrated in Figure 4.2. A first one extracts information about the environment of each residue. Then the second one is fed information about a pair of residues and it produces a cost matrix.

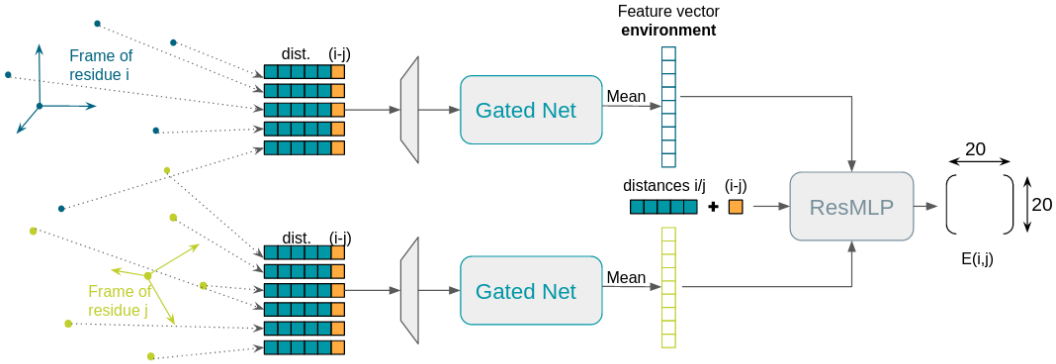


Figure 4.2: Overview of the neural architecture. First, a target residue and its neighbours are processed by a gated net to extract a feature vector informative on the environment of the target residue. Then, for each pair of residues, their environment are concatenated along with their relative features and given to a ResMLP which produces a cost matrix. The set of cost matrices forms the GM representing the protein design problem.

4.2.1 Extracting Environment Information

When we listed the desired properties of our protein backbone representation, we specified that the local environment of each residue should be taken into account as most of the interactions are local. We chose to represent the backbone as a set of pairs of residues, each pair being represented by invariant features describing relative position and orientation.

However, this is not directly informative about the environment of a given residue. Instead of putting this environment information on the representation explicitly - as it is done in graphs through the neighbourhood of a given node -, the

first part of the neural architecture is in charge of extracting it.

Neighbour Feature Extraction

For a given target residue i, all its neighbours within 10Å are extracted. The distance between C_α atoms is used. The features of each neighbour with respect to the target residue are computed and stacked into a tensor in which each line corresponds to a neighbour and each column is a feature. This tensor has shape $seq\_len \times nb\_ft$ where seq_len is the maximum number of neighbours (it is set to 45 as it is the maximum contact number at 10Å observed in the dataset) and nb_ft the feature dimension. If the target residue has fewer than seq_len neighbours, the vector is padded.

When using relative position features, the rotation of each neighbour with respect to the frame of residue i is expressed with a quaternion and its translation with 3D unit-vector, resulting in $4 + 3 = 7$ features. In addition, the distance is given using a GRBF encoding using 16 kernels, as well as the sequential information with a positional encoding of size 16 too. Finally, the contact numbers at 5 and 10Å and the cosine and sine of the dihedral angles ψ and ϕ are given for each residue of the pair, which represent respectively 4 and $2 * 4 = 8$ features. In total, the information of each neighbours with respect to the central residue is given by a vector of length 51.

When using distance features, the distances between each backbone atoms of the target residue and of each neighbour are computed. With the GRBF encoding, it results in $25 * 16 = 400$ features for each neighbour, to which are added the 16 features of the positional encoding.

Through the Neural Net

The whole protein backbone is processed into environment embeddings at once. To do so, the backbone is considered as a batch: the tensor of neighbours of each residue are stacked into a tensor of shape $n \times seq\_len \times nb\_ft$ processed in parallel. An input linear layer with $hidden\_dim$ neurons is applied first, followed by a gated-MLP (gMLP) [Liu et al., 2021]. This simple architecture has proven to be more efficient that Transformers in image application. In practice, we tested a GPT2 architecture with the same number of parameters and observed a drop in performance (see Annex C.1).

Through the gMLP layers, information on neighbours are considered and selected thanks to the gating mechanism. The gMLP output has the same shape as the input (*i.e.*, $n \times seq\_len \times hidden\_dim$). The contribution of each neighbour is then averaged into a tensor of shape $n \times hidden\_dim$ which contains the environment embedding vector of each residue. This vector is not interpretable *per se*, but it contains an internal representation of the central residue environment. We explored an alternative way to build the environment embedding, proposed in [Liu et al.,

2022]. We added a vector representing the target residue to the tensor representing its neighbours. After the tensor was fed into the gMLP, the processed target residue vector was used as environment embedding. However, it did not prove as efficient as simply averaging (see Annex C.1).

An Iterative Variant

Our intern Romain Gambardella suggested a variant architecture, illustrated in Figure 4.3. He replaced the input layer by a short resMLP and applied the same gatedMLP to obtain environment features. These features (before averaging) go through a second gMLP for several iterations. The input of the next iteration is the output of the previous one, concatenated with the first environment features. The output of the last iteration is averaged to form the environment vector.

It can be seen as gMLP blocks linked with residual connections. It is also similar to a Graph Neural Net, were each message passing step is done by one iteration through the second gMLP. However, standard GNNs do not have a gating mechanism like the one provided by the gMLP.

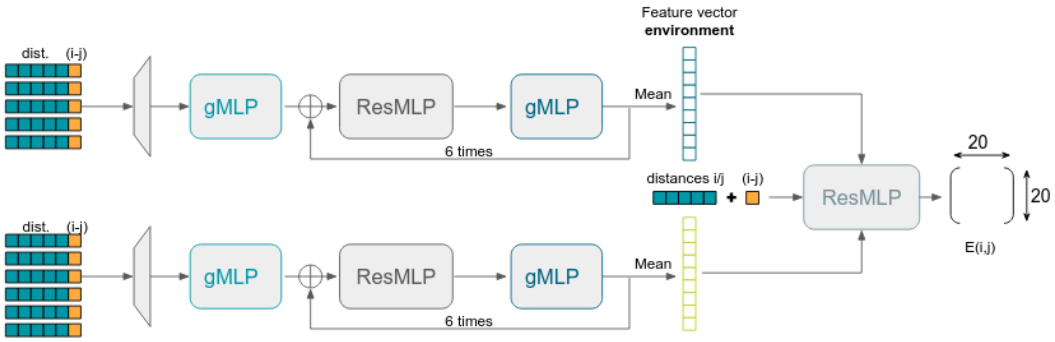


Figure 4.3: Iterative variant of the architecture. A new gMLP and a new ResNet are added to process environment. At each iteration, the current embedding and the features processed by the first gMLP are concatenated and fed to the second gMLP + ResMLP.

4.2.2 Predicting Pairwise Energy Matrices

The goal of the neural architecture is to learn a CFN whose cost matrices represent pairwise energy score functions. These score functions are (plus a constant) the negative logarithm of the probability of a sequence s given the input backbone B ($\log P(seq = s|struct = B)$) and they quantify the interaction within the input protein backbone. They are to be minimized to obtain the designed sequence(s).

A Graphical Model for Protein Design

A CFN (X, D, C) is composed of 3 elements. The variables X and their domain D are fixed by the input backbone. Indeed, the variables correspond to each position in the backbone, and each of them can take a value in the 20 canonical amino acids. Therefore, the only element that remains to be predicted by the neural architecture is the set of cost functions C.

As done by classical energy-based methods for Computational Protein Design (see Section 2.2.4), we restrict ourselves to pairwise energy terms. Therefore, our model is only able to consider 2-body interactions, by opposition to most DL-based methods that can represent many-body interactions. Even though considering many-body interactions is theoretically necessary [Kauffman and Weinberger, 1989], we did not find 2-body interactions limiting in practice.

For each pair of positions (i, j) in the input backbone, a matrix of shape 20×20 is predicted. It indicates how much it costs to simultaneously assign one residue identity to position i and another (possibly the same) to position j. The set of cost functions forms the predicted GM. For sequence design, it will be optimized in the exact same way as the traditional physics-based energies. This is why we sometimes abusively call the neural output an energy. However, it is just a score that cannot be directly physically interpreted.

The major difference between Effie and traditional energy functions is Effie being $-\log P(seq|struct)$ while the latter are $-\log P(seq, struct)$. Since $P(seq, struct) = P(seq|struct) \times P(struct)$, physics-based energy functions can assess the probability of the input backbone while Effie has no notion of what is a probable structure.

Neural Processing

We use a second neural net to predict these score matrices. It takes as input the environment feature vector of each residue of a given pair, concatenated with their relative features (either their relative position or a set of distances). In the case of multi-chain proteins, we added a bit indicating whether both residues are in the same chain.

According to the principle of parsimony, we used the simplest suited architecture to limit overfitting: a MLP with residual connections, which enable deeper architecture [He et al., 2016]. The size of the final layer is 400 so that the output can be reshaped in a 20×20 matrix.

We also explored the pertinence of using unary terms (results in the next chapter), *i.e.,* a term quantifying the cost of assigning one residue identity at a given position (independently of all the other positions). To predict them, we used another MLP to process the environment feature vectors into vector of shape 20 representing unary terms.

4.3 Training Schemes

The last thing left to do is to train the neural architecture we have just defined on our protein data. If the data and the neural architecture have to be crafted to tackle the specificity of protein structure data, training to predict a graphical model is a direct application of the pipeline we used to learn how to play Sudoku in Chapter 3. The only difference is the variation of size between protein instances, and it required only few adaptations.

In this section, we first describe the regularized loss we used and then the optimization process. Finally, we discuss the memory limitations we faced and how we dealt with them. All our code was written in Python using PyTorch version 11.10.2.

4.3.1 Loss Function

The Emmental-NPLL

The main purpose of developing a scalable loss for decision-focused learning problems such as learning to play Sudoku is to adapt it to protein design problems. Therefore, we used the Emmental-NPLL as our main loss.

The E-NPLL has only one parameter to tune: the number of holes h, *i.e.,* the number of variables to be masked when computing the pseudo log-likelihood. In the toy problem of Sudoku, the grid size was fixed, and therefore we use a fixed k. In the case of proteins, the problem size varies between instances. We adapt the E-NPLL by choosing a value of k proportional to the number of residues of the protein. Therefore, using the same notations as in Section 3.3.3, the E-NPLL becomes:

$$\text{E-NPLL}(\mathbf{y}) = -\sum_{Y_i \in \mathbf{Y}} \log(P^{N(\omega)}(y_i | \mathbf{y}_{-(\{i\} \cup \mathbf{M}_i)}))$$

where each $\mathbf{M}_i$ is a random subset of $\{1, \ldots, n\} \setminus \{i\}$ of size $k = \lfloor p \times n \rfloor$ with $p \in [0, 1]$.

Intuitively, the regular pseudo log-likelihood (*i.e.,* $k = 0$) consists in independently guessing the identity of each amino acid based on the other amino acids on the protein. The cost of each possible identity is turned into a probability using a softmax function. In this respect, the NPLL is similar to a task previously proposed to train neural networks for protein design: predicting one amino acid from its local environment [Zhang et al., 2020; Shroff et al., 2020]. Those methods use cross-entropy (CE) as the loss. The main difference between these two losses is the CE is directly applied to the neural net's prediction (independently on each variable), while the NPLL is a sum on every variables of marginal conditional probabilities computed based on the predicted GM.

When using the E-NPLL, only part of the environment of each residue is known. The E-NPLL was designed to tackle the regular PLL's inability to deal with logical information, and more generally with large energies [Montanari and Pereira, 2009]. It is not clear whether such information is present in the protein design problem. Yet, we found the E-NPLL leads to better performances than the regular PLL (see results detailed in the ablation study, Table 5.3).

An intuitive explanation can be proposed to explain the superiority of the E-PLL: in some cases, some amino acid assignations are indeed forbidden or highly unfavourable. For instance, if the environment of the target residue is such that only a small amino acid can fit in, large ones like tryptophan or tyrosine are associated with a high energy. Such redundant information can only be partially learned with the regular PLL.

Regularization

Like in the previous problem of learning to play Sudoku, we trained with a L1-regularization on the costs. The total loss is written as a term fitting the data (the E-NPLL) and a regularization:

$$\text{E-NPLL}(\mathbf{y}) = -\sum_{Y_i \in \mathbf{Y}} \log(P^{N(\omega)}(y_i|\mathbf{y}_{-(\{i\}\cup\mathbf{M}_i)})) + \lambda \sum_{i,j\neq i} \sum_{(a,b)\in D_i\times D_j} |N(\omega)[i,j](a,b)|$$

The L1-regularization favours sparse graphical models, which are appealing for two reasons. First, a sparse GM is likely to be easier to optimize. Second, it is a way to enforce the physics-based prior that far apart residues do not interact directly. Indeed, the NPLL alone will learn costs that tend to flatten with the distance but it has no reason to favour 0-cost.

When applying the $L1$-regularization, we indeed observed that the neural net learns a score that depends on the distance, as displayed in Figure 4.4. Above 10 Å, the score quickly decreases with the distance, and it is 0 beyond 20 Å. Using the regularization avoids the use of an arbitrary distance cut-off, which is common to both energy scoring function like Rosetta and neural-based methods, where the cut-off is used to define the neighbourhood (graph-based methods) or the local environment (voxel-based methods).

Variants

In some cases, we also predict unary terms $N(\omega)[i]$ that must be integrated into the loss as unary cost functions. They can be directly integrated to the E-NPLL computation by adding them to the unary costs computed given the rest of the variables (for a position i, $m_i = \sum_{j\neq i} N(\omega)[i,j](Y_i, y_j)$). We have also explored

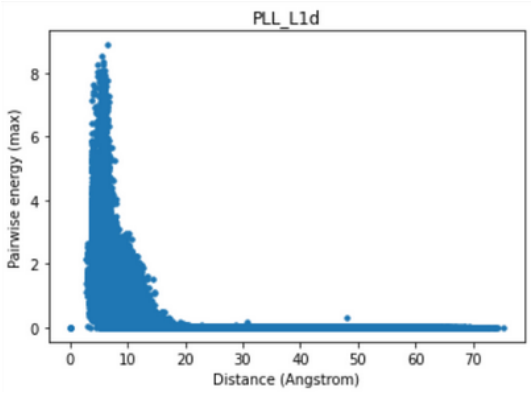


Figure 4.4: Variation of the cost of residue pairs with respect to their distance. Each dot corresponds to a pair of residue (i, j), whose associated cost is the maximum of the absolute valued of $N(\omega)[i, j]$.

multi-task learning where we want the unary costs to be predictive by themselves. In this case, we add to the loss the cross-entropy computed on the softmax unary costs. The total loss becomes:

$$Loss = \text{E-NPLL} + \text{CE} + \text{L1}$$

We also tried to include another prior knowledge: some amino acid types are more common than others. We plot in Figure 4.5 the amino acid distribution in the training dataset. In ML-terms, the prediction of one amino acid corresponds to an imbalanced classification problem. Thus, we implemented a balanced E-NPLL to favour rare amino acids by re-weighting each predicted cost by the square root of the inverse frequency of the corresponding class [Cui et al., 2019]:

$$m_{balanced}(i) = m(i) \times \frac{1}{\sqrt{freq(y_i)}}$$

where $freq(y_i)$ is the frequency of the amino acid type of y_i observed in the training set. Training under the balanced E-NPLL resulted in a predicted distribution of amino acid closer to the native one (in terms of Kullback-Leibler divergence: 0.91 with the balanced E-NPLL vs 3.05 without), but it decreased the NSR.

Similarly, we explored a "soft" PLL. The idea is to penalize confusion between similar amino acids less than between very different ones. Indeed, predicting an isoleucine instead of a leucine is likely to be better than predicting an arginin. Similarity between amino acids was measured using the BLOSUM62 substitution matrix [Henikoff and Henikoff, 1992]. The contribution of each residue to the NPLL, expressed as a unary term, is added to the similarity between the target residue and the current one. We use an addition instead of a multiplication because the PLL is similar to a logarithm of probability (energy).

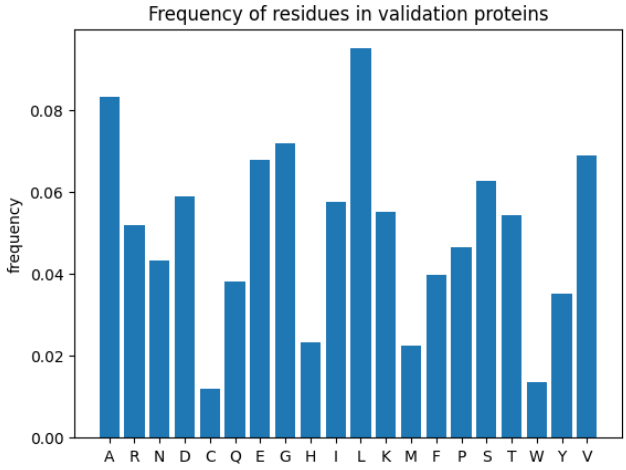


Figure 4.5: Histograms of the percentage of each type of amino acid in the validation dataset.

Finally, inspired by [Zaidi et al., 2022], who trained a denoising auto-encoder to predict molecular properties, arguing that is equivalent to learning a force field, we tried and learn a denoiser. Gaussian noise was added to each coordinate (independently), then a new MLP was connected to the gMLP output to predict the noise. The total loss was the sum of the NPLL and the mean square error on the predicted noise. As detailed in Annex C.2, none of these variants resulted in improved performances.

4.3.2 Optimization

Weight Initialization

The neural net's weight are initialized randomly, with a fixed seed to ensure reproducibility between trainings and compare models independently of the initialization.

Yet, a good weight initialization is important to prevent the outputs of the activation function of any layer from vanishing or exploding, which would reduce in respectively small or big gradients and result in slow convergence [Glorot and Bengio, 2010]. Since we used as activation functions the Gaussian Error Linear Unit (GELU) [Hendrycks and Gimpel, 2016], a differentiable variant of the ReLU, we chose the Kaiming intialization [He et al., 2015]. Bias of linear layers are initialized at 0. Moreover, in order to speed up training, a layer normalization is applied after each linear layer.

The Optimizer

We used Adam [Kingma and Ba, 2014], a variant of the stochastic gradient descent with adaptive moment estimation, as our optimizer. The neural net is trained with a weight decay of 10^{-3} to limit overfitting and an initial learning rate of 5.10^{-4}. All other parameters take default value (*i.e.*, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$). A L1 regularization of 10^{-4} is applied to the predicted costs.

We used a learning rate scheduling throughout training. It is based on the validation loss, that is defined as the percentage of amino acid correctly predicted given their environment. The learning rate is divided by 10 each time the validation loss decreases (*i.e.,* with patience 0) until it reaches 10^{-8}.

Adaptive Learning Rate

The way a protein instance is processed results in an unusual notion of batch. Indeed, even if the neural prediction is on a pair of residue, the NPLL is a sum over every variables. Therefore, a protein of n residues corresponds to $\frac{n(n-1)}{2}$ inputs and n outputs and so the size of the batch is difficult to define. Since proteins vary in size, the batch size varies as well throughout training.

In literature, it is usually recommended to choose the learning rate depending on the batch size bs, usually proportional to bs or to its square root [Smith et al., 2018]. Thus, we implemented an adaptive learning rate that depends on the size of the protein. For each training instance, the neural net is updated using as learning rate:

$$LR = LR_0 * n^{adapt}$$

with LR_0 the fixed learning rate and $adapt \in \{1, 0.5\}$.

Implementing this adaptive learning rate requires to be careful with potential interaction with the scheduler. Indeed, it is the basic learning rate LR_0 that is decreased when the validation loss decreases and not the current learning rate used by the optimizer.

We found that the adaptive learning rate has little impact on the single-chain dataset, which is composed of proteins up to 500 amino acids, but it becomes beneficial on the multi-chain dataset, with protein size up to $10,000$ residues.

4.3.3 Memory Limitations

The weights of the neural net are updated on each training protein structure independently, which represents an input of size $\frac{n(n-1)}{2}$ (with n the number of residues in the protein). Therefore, processing a single protein may not fit in the GPU memory. We first develop our model on a RTX-6000 card with 32Go of RAM, then we trained on a Nvidia A100 card with 80 Go memory.

Indeed, the neural net produces a cost matrix for each pair of residue in the protein, which corresponds to a space complexity of $O(n^2)$. To reduce it, we introduce a distance cut-off so that only close enough pairs of residues are processed. This cut-off is set at 15Å, which is more than classical energy-based or neural-based methods that used between 9 and 12Å. Combined with the L1 regularization, this still allows the neural net to learn the dependency between cost and distance as explained previously.

The distance cut-off makes it possible to process proteins up to around a thousand residues, but it is not enough when the size reaches several thousands. Reducing the size of the model or of the features (for instance, by reducing the number of Gaussian kernels used to encode distances) is not a good solution as it damages performances. Another idea is to process large proteins in several batches, but computing the PLL requires the complete GM, *i.e.,* all the pairwise costs.

We found a workable solution by changing the way the GM is represented. The first and easiest representation is a symmetric tensor W of shape $n \times n \times 20 \times 20$. The first 2 dimensions corresponds to the pair of positions and the last 2 are the domain sizes. Starting from a tensor filled with 0, all the pairs (i, j) with $i < j$ and within the distance cut-off are processed in parallel and used to fill the tensor representing the GM. The lower triangular part, corresponding to pairs with $j < i$ are filled by symmetrization.

This tensor is very sparse, especially for large proteins, as many pairs are not within the distance cut-off but their cost matrix (filled with zeros) is stored within the tensor W. A sparser representation is to store only the information of the closest neighbours. On the training set, the contact number (maximum number of neighbours) at 15Å is 128, so storing the 128 closest neighbours of each residue is enough to retrieve the complete GM. Therefore, we stored the GM as 2 tensors: one of shape $n \times 128 \times 20 \times 20$ contains the cost matrix of each residue and its closest 128 neighbours, and a second tensor of shape $n \times 128$ is used to remember the neighbours index.

From the sparse representation, it is easy to reconstruct the full representation W, which is more convenient to use. However, during training we only use the sparse representation, whose space complexity is $O(n)$ instead of $O(n^2)$. It set additional implementation challenges for both the symmetrization of the GM and the parallelized computation of the loss. With the sparse representation and the distance cut-off, all the proteins of the multi-chain dataset can fit on a single Nvidia A100 GPU card with 80 Go memory.

Conclusion

We have presented the whole methodology to learn Effie, the score function we will optimize to design new proteins. If the method we developed to learn graphical models, based on the Emmental-PLL, could be directly applied, the neural architecture and optimization had to be adapted to the specificities of protein data.

Effie is a general score function: it is learned once on all existing protein structures. It can be specialized to a given application problem during inference by adding constraints to the graphical model. Inference set new challenges that will be described in the following chapter, as well as the *in silico* and experimental validation of Effie.

We have learned Effie as the negative logarithm of the probability of the sequence given the structure, $-\log P(seq|struct)$. Optimizing this objective, as done by most DL-based design methods, does not imply that the target structure is optimal for the designed sequence, $P(struct|seq)$. Indeed, the sequence can have another global minimum, as noted by (Norn et al. [2020]). It has been argued that the objective $P(struct|seq)$ theoretically leads to increased protein stability and/or conformational specificity [Stern et al., 2023].

There are tight links between both objectives, as Bayes' theorem allows to express one wrt the other.

$$P(struct|seq) = \frac{P(seq, struct)}{P(seq)} = \frac{P(seq|struct) \times P(struct)}{P(seq)}$$

Since this objective is maximized on the sequence, $P(struct)$ can be eliminated from the objective. Therefore, using a tool predicting the probability of the sequence, such as a large Language Model trained on sequence databases [Elnaggar et al., 2021], $P(struct|seq)$ could be computed using Effie, similarly to what was done in (Stern et al. [2023]).

Bibliography

Anand, N. and Achim, T. (2022). Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*.

Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. (2000). The protein data bank. *Nucleic Acids Res.*, 28:235–242.

Chen, C., Zhou, J., Wang, F., Liu, X., and Dou, D. (2023). Structure-aware protein self-supervised learning. *Bioinformatics*, 39(4):btad189.

Cui, Y., Jia, M., Lin, T.-Y., Song, Y., and Belongie, S. (2019). Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9268–9277.

Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I., Courbet, A., de Haas, R. J., Bethel, N., et al. (2022). Robust deep learning–based protein sequence design using proteinMPNN. *Science*, 378(6615):49–56.

Eismann, S., Townshend, R. J., Thomas, N., Jagota, M., Jing, B., and Dror, R. O. (2021). Hierarchical, rotation-equivariant neural networks to select structural models of protein complexes. *Proteins: Structure, Function, and Bioinformatics*, 89(5):493–501.

Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., et al. (2021). Prottrans: Toward understanding the language of life through self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(10):7112–7127.

Fuchs, F., Worrall, D., Fischer, V., and Welling, M. (2020). Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in neural information processing systems*, 33:1970–1981.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919.

Hermosilla Casajús, P., Schäfer, M., Lang, M., Fackelmann, G., Vázquez Alcocer, P. P., Kozliková, B., Krone, M., Ritschel, T., and Ropinski, T. (2021). Intrinsic-extrinsic convolution and pooling for learning on 3d protein structures. In *International Conference on Learning Representations, ICLR 2021: Vienna, Austria, May 04 2021*, pages 1–16. OpenReview. net.

Ingraham, J., Garg, V. K., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Jiao, R., Han, J., Huang, W., Rong, Y., and Liu, Y. (2023). Energy-motivated equivariant pretraining for 3d molecular graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8096–8104.

Jing, B., Eismann, S., Suriana, P., Townshend, R. J. L., and Dror, R. (2021). Learning from protein structure with geometric vector perceptrons. In *International Conference on Learning Representations*.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Kauffman, S. A. and Weinberger, E. D. (1989). The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2):211–245.

Kenwright, B. (2012). A beginners guide to dual-quaternions: what they are, how they work, and how to use them for 3d character hierarchies. *Václav Skala-UNION Agency*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kozma, D., Simon, I., and Tusnady, G. E. (2012). Pdbtm: Protein data bank of transmembrane proteins after 8 years. *Nucleic acids research*, 41(D1):D524–D529.

Laine, E., Eismann, S., Elofsson, A., and Grudinin, S. (2021). Protein sequence-to-structure learning: Is this the end(-to-end revolution)?

Liu, H., Dai, Z., So, D., and Le, Q. V. (2021). Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34:9204–9215.

Liu, Y., Zhang, L., Wang, W., Zhu, M., Wang, C., Li, F., Zhang, J., Li, H., Chen, Q., and Liu, H. (2022). Rotamer-free protein sequence design based on deep learning and self-consistency. *Nature Computational Science*, 2(7):451–462.

López-Blanco, J. R. and Chacón, P. (2019). Korp: knowledge-based 6d potential for fast protein and loop modeling. *Bioinformatics*, 35(17):3013–3019.

McPartlon, M., Lai, B., and Xu, J. (2022). A deep se (3)-equivariant model for learning inverse protein folding. *bioRxiv*, pages 2022–04.

Montanari, A. and Pereira, J. (2009). Which graphical models are difficult to learn? In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

Norn, C., Wicky, B. I. M., Juergens, D., Liu, S., Kim, D., Koepnick, B., Anishchenko, I., Players, F., Baker, D., and Ovchinnikov, S. (2020). Protein sequence design by explicit energy landscape optimization. *bioRxiv*.

Shroff, R., Cole, A. W., Diaz, D. J., Morrow, B. R., Donnell, I., Annapareddy, A., Gollihar, J., Ellington, A. D., and Thyer, R. (2020). Discovery of novel gain-of-function mutations guided by structure-based deep learning. *ACS synthetic biology*, 9(11):2927–2935.

Sillitoe, I., Lewis, T., Cuff, A., Das, S., Ashford, P., Dawson, N., Furnham, N., Laskowski, R., Lee, D., Lees, J., et al. (2015). Cath: protein structure classification database. *Nucleic Acids Res*, 43(D1):D376–D381.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2018). Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*.

Sola, J. (2017). Quaternion kinematics for the error-state kalman filter. *arXiv preprint arXiv:1711.02508*.

Stern, J. A., Free, T. J., Stern, K. L., Gardiner, S., Dalley, N. A., Bundy, B. C., Price, J. L., Wingate, D., and Della Corte, D. (2023). A probabilistic view of protein stability, conformational specificity, and design. *Scientific Reports*, 13(1):15493.

Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A., and Kim, P. M. (2020). Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11(4):402–411.e4.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attentionis all you need. In *1st Conference on Neural Information Processing Systems (NIPS 2017)*.

Wang, J., Cao, H., Zhang, J., and et al. (2018). Computational protein design with deep learning neural networks. *Sci Rep*, 8(6349).

Zaidi, S., Schaarschmidt, M., Martens, J., Kim, H., Teh, Y. W., Sanchez-Gonzalez, A., Battaglia, P., Pascanu, R., and Godwin, J. (2022). Pre-training via denoising for molecular property prediction. In *NeurIPS 2022 AI for Science: Progress and Promises*.

Zhang, Y., Chen, Y., Wang, C., Lo, C.-C., Liu, X., Wu, W., and Zhang, J. (2020). Prodconn: Protein design using a convolutional neural network. *Proteins: Structure, Function, and Bioinformatics*, 88(7):819–829.

Zhou, Y., Barnes, C., Lu, J., Yang, J., and Li, H. (2019). On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753.

Chapter 5

Designing Proteins with Effie

Introduction

The main goal of this work is to learn a score function for Computational Protein Design (CPD). Chapter 4 has described in details how to learn our Energy Function Familiarly Introduced as Effie (Effie). Yet, it has not discussed how Effie is used for actual design. Effie has been designed to define a CFN that can be optimized by a discrete solver but full redesign CPD instances on large proteins are challenging. Standard methods alleviate tractability issues by simulated annealing, which may lead to increasingly non-optimal solutions [Simoncini et al., 2015]. Instead, we rely on the exact solver toulbar2 [Hurley et al., 2016] or a convex approximation of the problem that still offers some theoretical guarantees [Goemans and Williamson, 1995].

With the optimization, we are able to predict solutions of our learned problem, *i.e.,* the designed sequences. Since we placed ourselves within the Decision-Focused Learning framework, we will assess and tune the neural model by comparing our predicted sequences to the observed native sequences. The now-standard metric to evaluate DL-methods for CPD is the Native Sequence Recovery rate (NSR), even if it is imperfect, as we will discuss. As a first reference, 2 proteins with more than 30% sequence similarities are often considered to be from the same family and therefore share the same fold. Optimization details and hyper-parameter tuning is the object of Section 5.1.

Our selected models are then assessed on various design-related tasks and compared to existing approaches in Section 5.2. Part of the comparisons are versus existing pairwise-decomposable scoring functions for design, as our goal was to learn a better one. In particular, we compare with 2 energy functions from the standard RosettaDesign toolkit: ref2015 [Alford et al., 2017] and beta-nov16 [Pavlovicz

et al., 2020]. Both are all-atom and they predict $P(seq, struct)$. In those respects, they differ from Effie that is coarse-grained and predicts $P(seq|struct)$. The rest of our comparisons are versus pure DL-based methods, that are becoming more and more popular. They either directly map the sequence to the structure or they auto-regressively decode the sequence from the predicted conditional probability distributions. In both case, they bypass the need for optimization. Yet, as we will discuss, it is not necessarily an advantage for practical designs.

One recent method stands out from all the DL-based methods for CPD we described in Section 2. This is different with TERMinator (and its simplified version COORDinator) [Li et al., 2023], a pairwise-decomposable score function learned from data. These scoring functions also use a pseudo log-likelihood as a loss, even though we will show the Emmental-NPLL offers some boost in performances. One of the main difference with our approach reside in the optimization at inference, as they use simulated annealing. Since this approach is very similar to our work, it will have a dedicated place in our comparisons and discussion. Finally, Section 5.2 is also the place to discuss the utility of metrics evaluating DL-methods for design, and particularly NSR.

In fine, the only definite way to assess what has been learned by a DL-based method is to experimentally evaluate the designed sequences. We used Effie on 3 applied projects, some of them leading to experimental validations, as described in Section 5.3. We selected these projects for their biological interests, but they are also the occasion to illustrate additional constraints or properties that may occur in practice. In those cases, we will demonstrate the advantages of optimization-based design over sampled methods.

5.1 Optimizing the Learned Graphical Model

Once the neural network has been trained, it can be used to design new protein sequences from the coordinates of a backbone. It predicts the associated Graphical Model, which is then optimized to produce the designed sequence. This sequence is then compared to the observed solution (the native sequence) through the Native Sequence Recovery rate (NSR), *i.e.,* the percentage of similarity between both sequences.

In this section, we first describe how the inference is done, either with the exact solver toulbar2 or with the approximate and tractable method LR-BCD. Then, we use the NSR as the main metric to select the hyper-parameters of our neural architecture.

5.1.1 Inference

Exact Inference with Toulbar2

Starting from an input backbone, the neural net predicts cost matrices $E_{i,j}$ for each pair of residues at position (i, j). These costs are used in place of traditional physics-based or knowledge-based energy functions to define the optimization problem describing the Computational Protein Design problem:

$$\underset{i,j}{\operatorname{argmin}} \sum_{i,j=1}^{n} E_{i,j} = \operatorname{argmax} \log(P(seq|struct))$$

This optimization problem is solved using the discrete prover toulbar2 [Hurley et al., 2016] through its python interface PyToulbar2 version 0.0.0.2.

The problem is modelled as a Cost Function Network (CFN). Each residue at position $i \in \{1, \ldots, n\}$ in the backbone defines a variable with domain containing the 20 canonical amino acids. Each 20×20 matrix predicted by the neural net is used to define the binary cost function on the pair (i, j). Only the upper part of the matrix (corresponding to $i < j$) is needed to define these costs, hence the choice of imposing symmetric predictions. If the neural net makes predictions for single residues, they are used as unary cost functions.

To test the method, we perform a full redesign. However, some applications require to only design part of the protein (*e.g.,* the interface between several chains). In this case, the residues that are not redesigned are given as assigned to the solver.

This formulation of CPD being NP-hard [Pierce and Winfree, 2002], tractability issues arise from the optimization. We tried and limit them by predicting sparser matrix thanks to the L1 regularization. We also limit the resolution of toulbar2 to 2 digits after decimal point. Yet, exactly optimizing large proteins (more than one or two hundred residues) is intractable. Therefore, we decided to use approximate

solvers. The first one was toulbar2 with a limited number of allowed backtrack for some control on the trade-off time/solution quality. However, there are no guarantee on the quality of the solution found when the search is stopped at the end of the allowed backtracks.

Optimizing Large Proteins with LR-BCD

For a more tractable optimization with theoretical guarantees, we rely on the LR-BCD method that solves a convex relaxation of the MAP-MRF problem and uses stochastic rounding to provide solution [Durante et al., 2022].

LR-BCD has 3 parameters to be chosen: the number of iterations, the rank and the number of roundings. Iterations and rank control trade-off between speed and closeness to optimality of the convex relaxation. Computations are done with continuous variables, which are then brought back to discrete domains by several rounding procedures, whose number is controlled by the third parameter. The combination of convex relaxation with stochastic rounding offers guarentees in terms of distance to optimality [Goemans and Williamson, 1995]. Each rounding procedure is followed by a local optimization to possibly further improve the quality of the output solution.

To compute the NSR of the sequences designed with LR-BCD, we tested 3 methods. We averaged the NSR over all the designed sequences or we selected the sequence with the lowest predicted energy. We also tried an intermediate method where we averaged the NSR of each sequence weighted by the softmax of the energy. As detailed in Annex B, selecting the best sequence gives the best NSR. Since the rounding is stochastic, the designed sequence may change between runs.

We experimentally chose each parameter of LR-BCD by fixing the others and make it vary. All the experimental details are in Annex B. We found the rank and the number of iterations do not have a large impact (see Figure B.1b and B.1a respectively). We set the number of iterations to 5 and we use the option -4 for the rank. The number of rounding impacts the variability of solutions between runs (see Figure B.1c). Using 50 roundings instead of 20 reduces the energy variability, but further increasing to 100 roundings does not seem to improve further. Therefore, we predicted 50 sequences and chose the best one in our experiments.

To ensure the quality of LR-BCD solutions, we compared them with the solutions, guaranteed to be optimal, obtained by the exact solver toulbar2. We fully redesigned with both methods all the proteins from the monomeric dataset, which are shorter than 150 residues. Solutions found by LR-BCD are in average at 0.56% of the optimal Effie score, so LR-BCD provide protein sequences close to the optimum. We also compared the sequences designed by both solver with native sequences in terms of similarity. Results are plotted in Figure 5.1. Sequences by LR-BCD have a similar NSR to those obtained by exact optimization.

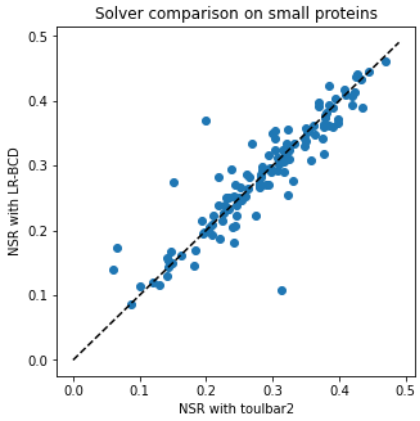


Figure 5.1: Similarity to the native sequence (NSR) of sequences designed with toulbar2 or LR-BCD.

Finally, we compared LR-BCD with another approximate solver: toulbar2 with no backtrack allowed. Both gives similar results but LR-BCD is faster (see Figure B.2). Therefore, we use LR-BCD as default solver for *in silico* validation. It can optimize proteins up to $1,500$ residues. Our applied projects focus on small proteins or partial redesign, so we could use toulbar2.

5.1.2 Hyperparameter Tuning

The main metric to assess DL-based design pipelines is the *Native Sequence Recovery* rate (NSR), *i.e.,* the sequence similarity between the native and designed sequences. Even though it has many drawbacks, as discussed in Section 2.3.3, we used it as the main criteria to select our neural model and tune its hyper-parameters.

Hyperparameter List

The hyperparameters of our architecture are:

- Optimization:
  - Learning rate: initial value ($5 * 10^{-4}$), scheduler (divide the LR by 10 each time the validation loss decreases, with patience 0)
  - Algorithm: Adam [Kingma and Ba, 2015] with a weight decay of 10^{-3}. Other values take default parameters
- Architecture
  - gMLP: sequence length (48), input layer width or embedding dimension (128), output dimension (128), width (multiplying factor of the input dimension, here 4), number of layers or depth (12)

  - ResMLP: number of blocks (3) and block size (2), hence resulting in 6 layers, width (128)

- Features
  - Number of kernels and of positional embeddings (both 16)
  - Distance cut-off between pairs of residues (15) and number of neighbours (128)

- Loss
  - E-NPLL with $p = 0.3$. Only the binary terms are used.
  - Regularization on the learned cost function: L1, with a regularization term of 10^{-4}

The indicated values are those on the single-chain dataset. None of the variant losses we tried (balanced PLL, soft PLL, cross-entropy on unary terms, denoiser) led to improved performances.

The values of each hyperparameters have been chosen by fixing all parameters saved one and training a model whose performances was assessed by its NSR on the validation set. In the case where two models had similar NSR, we applied parsimony and chose the simplest model (*e.g.,* with fewer parameters or more regularization). Details of the choice of each hyperparameter are described in Annex C.

An Iterative Variant

A variant architecture was proposed by our intern. The embedding dimension is twice as large (256). A first ResMLP with 3 blocks and a first gMLP with 3 layers (instead of 12) and the same the width ($2 * 256$) is applied. Then a second ResMLP and a second gMLP, with same shape, are iteratively applied. Finally, the resulting environment embedding is processed by a third resMLP (3 blocks of size 2 and width 256) to produce cost functions. All the features and optimization parameters are kept the same as for the non-iterative version. It contains 3.4M parameters, vs 2.6M for the previous model.

Extension to Multi-chain Proteins

Multi-chain proteins are larger (in our dataset, up to $10,000$ residues instead of 500) and contains both intra and inter-chain interactions. To enable Effie to capture both interactions, we hypothesized that an increase in the neural net capacity was required. We verified it in Annex C.3, and we made some change in the architecture: the gMLP was deepened to 15 layers, and the resMLP was both deepened to 10 blocks and enlarged to a hidden dimension of 256.

We also started from a lower initial learning rate ($5 * 10^{-5}$) and we found that the adaptive learning rate was now beneficial (see Table C.9). More precisely, we changed the learning rate for each protein such that it was proportional to the square root of the number of variables (akin to the batch size). Training and testing on multi-chain proteins led to better residue-level and protein-level predictions than on the single-chain set (Table 5.1). For memory issues, only multi-chain proteins shorter than $1,500$ amino acids were considered.

Dataset	Single-chain	Multi-chain
Accuracy	46%	57%
NSR (median)	42.8%	47.7%

Table 5.1: Comparison of performances obtained on the single-chain and multi-chain dataset. Accuracy is a per-residue metric assessed on predicting one residue given the rest of the protein, while NSR is a per-protein metric.

For practical designs, we trained additional models. It is suggested in ProteinMPNN [Dauparas et al., 2022] that adding a small Gaussian noise to the coordinates of training proteins helps improving forward folding. We trained additional models, with noise with standard deviation of 0.2, 0.14 or 0.02 Å. Since our applications (in Section 5.3) deal with soluble proteins only, we trained these noised models by excluding all transmembrane proteins. The list of these proteins is included in the dataset.

Effie Versions

Throughout the thesis, different versions of Effie were developed. Some application were started before the latest versions were developed and therefore they use former version. All the existing versions are summarized in Table 5.2.

		Quat.	Distances		
Dataset	Noise	V1	V2.1	V2.2	V3
Single	0	x	x	x	x
Multi	0		D	D	x
Multi	0.02			D	x
Multi	0.14		D		
Multi	0.2			D	x

Table 5.2: Versions of Effie mentioned in this work. D means the version has been used for practical design.

Effie versions differ from architecture and training settings. V1 is the version using quaternions, while all others use distances. V2 is our current default version; V2.1 and V2.2 corresponds to the same architecture, but the V2.2 is optimized in memory (as described in described in Section 4.3.3). By opposition to V2.2, V2.1 cannot been trained on proteins longer than $3,000$ residues due to memory limitation, which represents only 3.6% on proteins but 23% of residues pairs within 15Å. Still, most of the time, V2.1 and V2.2 lead to similar performances and therefore the sub-version is not specified. V3 is the iterative variant. As it is very recent, it has not been applied on practical designs yet. They can be trained on the single-chain or the multi-chain datasets, with optional noise of the backbones. For instance, Effie V2.2-multi-0.2 is default version for design. It is the architecture optimized in memory, trained on the multi-chain dataset by adding a Gaussian noise with standard deviation of 0.2Å to input backbones.

Ablation Study

We made sure that all the major components of our architecture are useful through an ablation study displayed in Table 5.3. Results were obtained with Effie V2.1 on the single-chain test set.

Model	NSR
Effie	**42.8%**
No ensemble learning ($p = 0.3$)	42.3%
Regular PLL ($p = 0$)	41.2%
No environment information (*i.e.*, no gMLP)	33.5%
No sequence information (feature $i - j$)	41.2%
No distance cut-off at 15Åbetween residue pairs	40.3%
No L1 regularization on learned costs	41.9%

Table 5.3: Ablation study performed on the single-chain test set.

Our final model is an ensemble of three models, trained with different values of the E-NPLL (respectively $p = 0.1$, $p = 0.3$, $p = 0.4$), which allowed a gain of 0.5% NSR when compared to a single model. Interestingly, even though we developed the E-NPLL to deal with logical information, it also proved beneficial in the case of protein data as it lead to a 1.1% increase in NSR. We hypothesize this improvement comes from the existence of high energies in CPD problems, that cannot be estimated with the regular NPLL, as explained in Subsection 3.3.1.

Removing the dedicated neural net to extract the neighbour information of each residue is the ablation that lead to the largest drop in performances, which is coherent with physics-based intuition. The distance cut-off on pairs of residue is beneficial

in terms of NSR, but also in terms of memory usage and optimization time. It can be seen as an inductive bias, as far apart residues do not interact. Similarly, the L1 regularization, by encouraging the predicted costs to be sparse, speeds up optimization while improving NSR.

5.2 Assessing Effie *in silico*

As we are in an unsupervised setting, there is no ground-truth energy. Thus, the quality of the learned cost functions cannot be assessed directly. Instead, we rely on auxiliary tasks.

The first tests were done at the residue level to assess whether basics physics-chemical properties are recovered. We then performed full design on test structures and we compared the predicted and native sequences. The quality of the designed sequences was also assessed by forward folding, *i.e.,* their structure was predicted *in silico* and compared to the target backbone. Finally, we assessed the limits of Effie on a structure quality prediction task.

5.2.1 Recovering Natural Amino Acid Properties

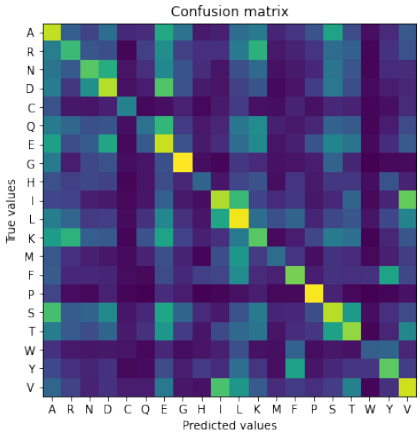
At residue-level, we made two tests : predicting one residue given its environment and predicting the effect of a single mutation. The results suggest that Effie has learned some insights about what a functional protein is.

Predicting One Masked Residue

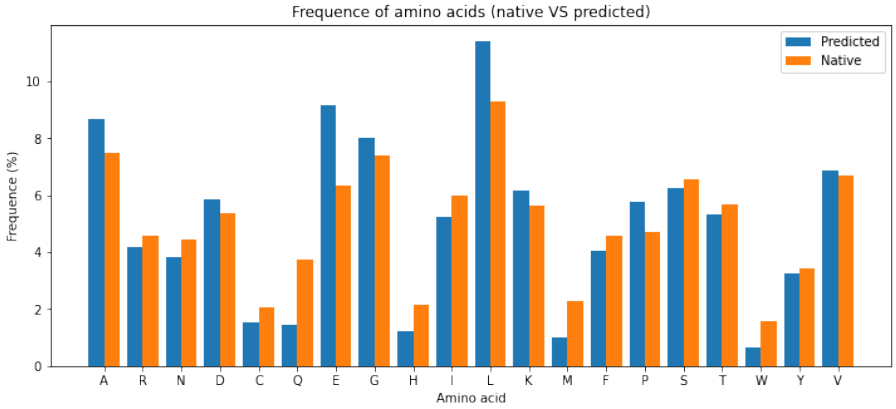
The first task we assessed Effie (V2-single) on is the prediction of one masked residue while the rest of the protein is known. This task is included into its training objective, thus it performs well at it: 46.2% of the residues of the test proteins of the single-chain dataset are correctly predicted (missing residues are excluded for the task). Moreover, we estimated core and surface residues based on the number of neighbouring residues (respectively more than 25 and less than 15) and we found that amino acids at the core of the protein are better predicted (62.6% are correct) than at the surface (39.2%). This was expected as core residues are known to be more constrained than surface ones.

As expected, per-residue accuracy is better on the multi-chain dataset (see Table 5.4). The iterative variant also considerably improves this metric. These results cannot be directly compared with CNN-based approaches predicting one residue from its environment (for instance 57% accuracy in [Anand et al., 2022]). Indeed, they use all-atom information (so the exact conformation of environmental side chains are known) while we only have coarse-grain representation.

When Effie makes mistakes, it tends to confuse residues with similar physic-chemical properties. This can be seen on the confusion matrix (Figure 5.2a). For instance, aspartic acid (D) is often mistaken for glutamic acid (E), which are both negatively charged. Similarly, leucine (L) and isoleucine (I) are often switched. On the contrary, amino acids with very specific properties, such as glycine (G), proline (P) and cystein (C) are almost always correct. When compared with the BLOSUM62



(a) Confusion matrix



(b) Distribution of predicted and ground truth amino acid identity.

Figure 5.2: Effie's predictions on the task of predicting one amino acid from its environment.

substitution matrix [Henikoff and Henikoff, 1992], Effie's predictions are similar to the native residue (they have a positive BLOSUM62 score) in 74.6% of cases.

Finally, Effie's predictions on a single residue mostly recover the natural amino acid propensity (see Figure 5.2b). Effie seems to slightly over-favour common residues such as leucine (L), alanine (A), glycine (G) and glutamic acid (E) while rare residues like tryptophan (W), cystein (C) or histidine (H) are ralerly predicted. This is likely to be a direct consequence of the dataset being unbalanced. We compared both predicted and native distributions in terms of Kullback-Leibler divergence (lower is better): with default training, the KL divergence is 3.05 on the single-chain dataset. When using the balanced E-NPLL (described page 144), it drops to 0.91, but full-design metrics deteriorate.

Model	Test set	Accuracy
Effie (V2)	single-chain	46.2%
Effie (V3)	single-chain	51.2%
Effie (V2)	multi-chain	54.1%
Effie (V3)	multi-chain	59.5%

Table 5.4: Per-residue accuracy of various version on Effie, computed on either the single-chain or the multi-chain test set.

Predicting the Effect of a Single Mutation

The ability to predict one residue given its environment is tightly linked to the ability to predict the effect of single-point mutation, which in turn can enable design. For instance, the activity and thermostability of an enzyme degrading PET were enhanced by identifying the native residues that were the least suited to their environment and mutate them [Lu et al., 2022].

Score function	Accuracy	AUROC
Effie (V2-multi)	78.3%	**0.66**
Effie (V3-multi)	77.2%	0.65
beta-nov16	38.0%	0.44

Table 5.5: Comparison between Effie and beta-nov16 on the atom3d benchmark.

Even though it was not explicitly trained for it, we assessed Effie's ability to do 0-shot prediction of the effect of a single mutation on two benchmarks. The first one is the Mutation Stability Prediction task of Atom3D [Townshend et al., 2020]. The dataset is based on the SKEMPI database [Jankauskaitė et al., 2019] of binding free energy changes upon mutations within interfaces of complexes. Single-point mutations were collected and mutated structure were obtained via molecular modelling. As we did not retrained Effie for the task, we only used the test set. The task is a binary classification: predicting whether a mutation at the interface will stabilize the complex. We scored each native/mutant couple with either Effie (V2-multi), Effie (V3-multi) or beta-nov16 [Pavlovicz et al., 2020]. If the mutant has a lower energy score, the mutation is classified as stabilizing. We also computed the AUROC, the recommended metric for this benchmark. On both metrics, both Effie versions largely outperforms betanov16 (see Table 5.5). Interestingly, the V2 performs slightly better than the V3, even though the V3 is far superior in previous task (Table 5.4).

The second benchmark [Rocklin et al., 2017] is composed of 10 *de novo* mini-

proteins with various folds. Each have 775 variants whose stability has been experimentally assessed. The goal is to predict the stability of each variant. The metric used is the Pearson correlation between the stability score and the predicted energy. We compared Effie with 3 design methods presented in Section 2.3.3, all trained on the single-chain dataset using : Structured Transformer (Str-Trf) [Ingraham et al., 2019], the Geometric Vector Perceptron (GVP) [Jing et al., 2021] and COORDinator [Li et al., 2023]. First, all DL-based methods largely outperformed beta-nov16 that reached an average correlation of 0.14. As displayed in Table 5.6, Effie V2, GVP and TERMinator lead to similar results. Once again, Effie (V3) is slightly outperformed by the default version.

Fold	Str-Trf	GVP	COORDinator	Effie (V2)	Effie (V3)
EEHEE 37	0.47	**0.60**	0.53	0.59	0.55
EEHEE 1498	**0.45**	0.38	0.37	0.39	0.39
EEHEE 1702	0.12	0.26	0.23	0.25	**0.27**
EEHEE 1716	0.47	0.54	**0.56**	0.54	0.50
HEEH 779	0.57	**0.58**	0.48	0.57	0.55
HEEH 223	0.36	0.47	0.51	**0.52**	0.40
HEEH 726	0.21	0.22	**0.23**	0.18	0.15
HEEH 872	0.23	0.33	**0.36**	0.33	0.32
HHH 134	0.36	0.45	**0.46**	0.42	0.35
HHH 138	0.41	0.48	**0.49**	0.42	0.41
Average	0.37	0.42	**0.44**	0.42	0.39

Table 5.6: Comparison of 4 design methods on stability prediction of *de novo* mini proteins. The fold is indicated with E for β-sheets and H for α-helices. All methods were trained on the single-chain dataset.

5.2.2 Full Redesign

After training and assessing Effie on residue-level tasks, we tested it on the design of full proteins. From an input backbone, the neural net predicts the set of Effie score functions, which are optimized to produce a protein sequence.

We compare first with other methods based on a pairwise-decomposable score function, which are similar to ours. Then we compared to non-decomposable methods, which encompasses most of DL-based design approaches. The first group of methods (including ours) is limited to 2-body interactions while the second group can represent many-body interactions. In both cases, comparisons are made based on the NSR metric. Finally, we highlight the possibility of adding *a posteriori* knowledge to decomposable approaches without retraining.

Comparison with Pairwise-decomposable Methods

Since our main goal is to learn a pairwise energy scoring function that is better than existing one, we first compared with the Rosetta design pipeline [Park et al., 2016] (results are extracted from [Ingraham et al., 2019]). Since the comparison was done on the monomeric proteins from the single test, which are shorter than 150 amino acids, we optimized Effie (V2-single) with both toulbar2 and LR-BCD. As displayed in Table 5.7, both optimization schemes lead to very similar results in terms on NSR, and largely outperform Rosetta.

Metric	LR-BCD	toulbar2	Rosetta
NSR (median)	29.4%	**29.8%**	17.8%
Optim.	Convex approx.	Exact	SA

Table 5.7: Comparison of the performance of the approximate solver LR-BCD and the exact solver toulbar2 when optimizing Effie (single) on small proteins. Additional comparison is made with the standard Rosetta design pipeline, based on simulated annealing (SA).

To the best of our knowledge, there are 2 other DL-learned pairwise score functions for design, both proposed by [Li et al., 2023]: COORDinator, based solely on the structure coordinates (like us), and TERMInator, that additionally used costly-to-computer Tertiary Repeating Motifs (TERMs). They differ from our approach as they design sequences by simulated annealing (like Rosetta) and not by exact or convex optimization. We compared them with Effie (V2-single) and Effie (V3-single) in Table 5.8, all 4 score functions being trained and tested on the same dataset. Effie (V2) outperformed COORDinator with a margin of 2.5 points. It is only slightly above TERMinator, but Effie does not require TERMs inputs, that take 4 minutes per residue to compute (SI of [Li et al., 2023]). Effie V3 considerably outperforms all other scores.

Metric	TERMinator	COORDinator	Effie (V2)	Effie (V3)
NSR	42.4%	40.3%	42.8%	**47.9%**

Table 5.8: Comparison between learned score functions for design on the single test set. Median NSR is given.

Another usual metric to assess a fully-redesigned sequence is perplexity, defined as the probability of a residue identity given previous residues:

$$-\exp(-\frac{1}{t}\sum_{i}^{t}\log P(x_i|x_{<i}))$$

with n the length of the sequence. We cannot compute this perplexity as it would require to estimate the partition function of the probability distribution resulting from the CFN, a #P-hard task. Instead, we defined a pseudo-perplexity, based on the negative pseudo log-likelihood instead of the negative likelihood :

$$-\exp(-\frac{1}{t}\sum_{i}^{t}\log P(x_i|x_{-i}))$$

The pseudo-perplexity on the single-chain test set id 5.11 for Effie V2 and 4.46 for the V3. It means the model is as confused as if it had to make a choice between 5 residue identities (instead of 20 if randomly chosen).

Comparison with Non-decomposable Methods

Non-decomposable approaches use only Deep Learning, and no discrete optimization compound. They present the advantage of allowing the neural net to consider high-order interactions. Most of them auto-regressively decode the sequence by predicting the probability distribution over one amino acid given the structure and the already-decoded sequence, and sample the next residue from this distribution. This includes Structured Transformer (Str. Trf) [Ingraham et al., 2019], the Geometric Vector Perceptron (GVP) [Jing et al., 2021] and ProteinMPNN [Dauparas et al., 2022]. Other methods, like PiFold [Gao et al., 2023] directly maps the input backbone to an output sequence.

Once again, all these methods have been trained and tested on the single-chain dataset and therefore they can be directly compared, as done in Table 5.9. Note that the score reported for ProteinMPNN differs from the one presented in the original paper. Indeed, ProteinMPNN authors directly assigned residues with missing coordinates to the native ones. Instead, we report the NSR when missing residues are removed [Gao et al., 2023], as done by other methods.

Approach	Str. Trf	GVP	ProteinMPNN	PiFold
Type	AR	AR	AR	Direct map
NSR	36.4%	40.2%	45.9%	51.6%

Table 5.9: NSR on the single-chain dataset achieved by pure DL- non-decomposable models. They either decode the sequence auto-regressively (AR) or directly map the input backbone to the sequence.

If the most recently proposed method PiFold improve over Effie in terms of NSR, practical design is not about recovering the native sequence. Instead, additional properties on the designed sequence often need to be enforced, which cannot be done with directly-mapping methods like PiFold. With auto-regressive models, some

properties can be enforced, but it is not straight-forward. For instance, designing diverse sequences requires to increase the sampling temperature, which mechanically degrades the quality of the sequences. With an exact solver like toulbar2, the trade-off between quality and diversity can be optimized and controlled as it is possible to find the best possible sequence that differs from the optimal sequence by a given number of mutations [Ruffini et al., 2019].

We will illustrate some other specific tasks that can only be done by *a posteriori* conditioning and biasing. Additional knowledge can be considered *a posteriori* to bias Effie's probability distribution. Conditioning is done trough additional constraints. Examples are given in our applied projects in Section 5.3.

Additional Insights

The NSR, by comparing with the native sequence alone, does not take into account the fact that several sequences can adopt the same fold. Therefore, a sequence that indeed folds onto the target structure may have a poor NSR score. To take this degeneracy into account, we additionally computed the native sequence similarity rate (NSSR), defined based on the BLOSUM62 [Henikoff and Henikoff, 1992] substitution matrix. If a predicted residue has a positive similarity score with the native one, it will be counted as correct for the NSSR. With Effie V2, we reached a NSSR of 72% on the single-chain dataset 76.2% on the multi-chain, suggesting again that the substitutions made by Effie are chemically and physically plausible.

Model	NSSR (single)	NSSR (multi)
Effie V2	72%	76.2%
Effie V3	76.4%	79.3%

Table 5.10: Native sequence similarity rate (NSSR), on 2 architectures and on 2 test sets (single-chain and multi-chain).

In order to be more confident in using Effie for practical designs, we further assessed the sequences that were poorly designed, defined as proteins with a NSR$<$ 30. In the multi-chain test set, there were 4 of them: 3 are coiled-coils (PDB ids 6FPR, 1GD2 and 4CFG), *i.e.* bundles of α-helices, which is similar to what was observed for TERMInator [Li et al., 2023]. The last one (5TS4) is a *de novo* protein with an experimentally resolved idealized structure. Outside of these very particular cases, the NSR is always above 30, which is considered the threshold above which two proteins are likely from the same family and share the same fold.

Adding Knowledge *A Posteriori*

With our method, any knowledge can be added *a posteriori* to bias the prediction without retraining as long as it can be expressed in the CFN framework. We

illustrate this capability on extending Effie to transmembrane proteins without re-training.

We use the multi-chain dataset that contains a total of 3498 transmembrane proteins. Membrane residues are identified using TMbed [Bernhofer and Rost, 2022], a DL method based on a large language model to predict the membrane residues of an input sequence. It has a very low false-positive rate (fewer than 1% of non transmembrane proteins have a residue predicted as membrane). We used TMbed in default configuration to predict the membrane residues of all the dataset. Those residues are uncommon (0.7% of all residues in the multi-chain dataset).

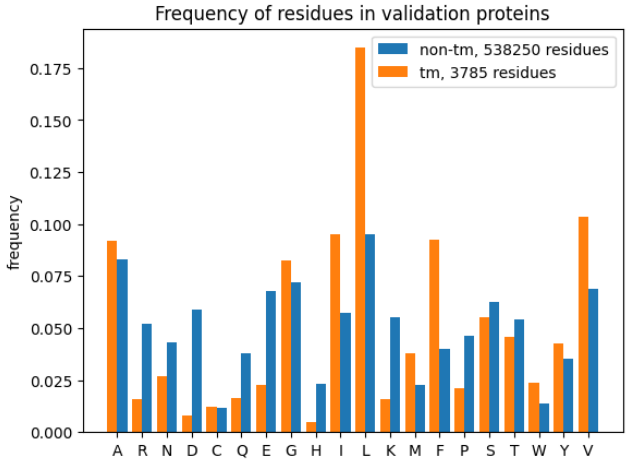


Figure 5.3: Distribution of amino acids within membrane (orange) and out of membrane (blue), computed on the proteins of the multi-chain validation set. Tm stands for transmembrane.

A cell membrane is composed of lipids, whose polar heads are at the surface of the membrane (in contact with the solvent) and the long apolar tails are within the membrane. Therefore, the part of a protein within the hearl of the membrane is in an apolar environment instead of a polar one. Therefore, the distribution of amino acids within a membrane changes from the distribution within soluble parts, as shown in Figure 5.3. For instance, charged residues (both negatively — D and E — or positively — K, R H —) are much less common in a membrane.

Effie does not perform as well on predicting membrane residues: given the environment, 54.5% of soluble residues are correctly predicted by the V2-multi version, vs only 46.0% of transmembrane residues. We hypothesized that the distribution being different is a useful inductive biais to predict them better. We *a posteriori* added a unary term F_{tm} to membrane residues to bias the distribution. This term is negative when an amino acid type is more common within membranes to encourage

its prediction.

$$F_{tm}[i] = \log(\frac{\mathcal{D}_{soluble}[i]}{\mathcal{D}_{tm}[i]})$$

with $i \in D$ a type of amino acid, $\mathcal{D}_{tm}$ (resp. $\mathcal{D}_{soluble}$) being the distribution of transmembrane (resp. soluble) residues and so $\mathcal{D}_{tm}[i]$ is the frequency of type i in transmembrane residues.

We fully redesigned the transmembrane proteins of the test set (shorter than $1,500$ residues) either with no additional information or with the unary transmembrane term. We observed a substantial gain in median NSR (see table 5.11). Moreover, for each protein, the NSR when adding unary terms is at least as good as the NSR with the default model. Therefore, we successfully biased Effie *a posteriori* to adapt it to score transmembrane residues.

Model	NSR (median)
Effie	36.0%
Effie + unary tm	37.9%

Table 5.11: NSR on the full redesign on transmembrane test proteins, with Effie (multi) or with the same model and additional transmembrane (tm) unary terms.

An auto-regressive DL-model like ProteinMPNN [Dauparas et al., 2022] can be biased as well. Indeed, since it predicts the distribution over a target residue (knowing the par of the sequence that have already been decoded), this distribution can be biased to favour transmembrane amino acids as we did. However, the decision made for the target residue is independent of future choices, which is likely to result in non-optimal sequences. Moreover, it is only possible to control the choice over single residues, not pairs or more.

Biasing the model by adding knowledge on transmembrane residues *a posteriori* indicated that this knowledge is a good inductive bias. Therefore, we tried to train a model with one additional feature indicating whether a residue is within the membrane. Surprisingly, the resulting model learned to ignore this feature: the prediction is the same when membrane residues are identified or not. We hypothesized this behaviour could be due to imbalance between in and out-membrane residues (0.7% of residues are within a membrane in the multi-chain dataset). Penalizing more errors on membrane residues (from 10 to 100 times) did not alleviate this issues. We then hypothesized that this feature being per-residue (and not on residue pairs), introducing unary costs to take it into account may be beneficial. Yet, the resulting model did not result in better predicting membrane residues or better designing trans-membrane proteins. Further investigating this behaviour may lead to improving the overall model.

5.2.3 Forward Folding

The CPD task aims to recover an input backbone structure (with optional additional properties in practice). Therefore, a straight-forward way to assess the designed proteins is to verify they fold back onto the target structure. This evaluation is called *forward folding*. If experimental characterization is long and costly, the recent advances in protein structure prediction enable a fast and quite reliable evaluation. For instance, AlphaFold2 reaches an experimental accuracy on monomeric proteins, when used with MSA and template [Jumper et al., 2021].

Most of the design methods proposed after CASP14 use AlphaFold2 for forward folding, except when they have already used it for generating data or within the design pipeline [Hsu et al., 2022; Goverde et al., 2023]. In these cases, an independent baseline such as TrRosetta [Yang et al., 2020] or ESMFold [Lin et al., 2023] is used. So far, there is so far no consensus on the metric to use. If the alignment between the predicted and the target structure is interesting, for instance measured with the Root Mean Square Deviation (RMSD), assessments are often based on AlphaFold2's confidence metrics. ProteinMPNN [Dauparas et al., 2022] used the mean pLDDT on several designs, TERMinator prefered the pTM score [Li et al., 2023] and DLbinder design [Bennett et al., 2023] chose the pAE.

Moreover, AlphaFold2 is not always used in the same conditions: ProteinMPNN uses it in single-sequence mode while TERMinator gives an MSA as additional input. All these differences in usage make comparison between methods difficult, all-the-more-so results are often given as plots rather than tables. We chose to use the same setting as ProteinMPNN: single-sequence mode with 3 recycles. Indeed, we want to test our design method independently of the number of homologous proteins of test proteins. We run design on LocalColabFold [Mirdita et al., 2022] version 1.5.2.

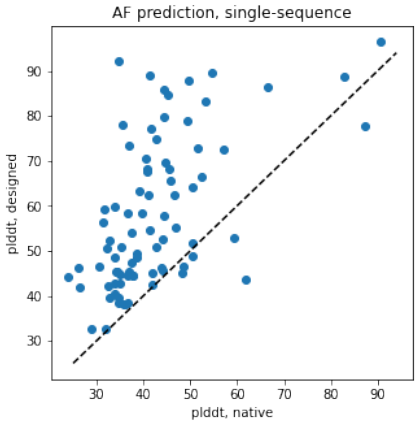


Figure 5.4: The pLDDT of the designed vs the native sequences. Each point represent a backbone. Done with Effie multi, on the multi-chain test set.

We first compared the confidence of AlphaFold2 on folding designed sequences VS their native counterpart. The performance of AlphaFold2 from a sequence only is usually poorer, because it lacks all the information derived from evolutionary data. Here we observed in Figure 5.4 that pLDDTs are almost always better for the designed sequences, sometimes with an important gap. It means designed sequences more strongly encode structure than native sequences, which is coherent with observation on ProteinMPNN [Dauparas et al., 2022]. We agree with their explanation that evolution does not optimize solely for stability (other properties linked to survival in the environment are optimized as well), leading to weaker sequence-structure mapping.

We then compared the different versions of Effie on our multi-chain test set (93 soluble proteins). Some versions were trained with coordinates blurred with a Gaussian noise because it was previously observed [Dauparas et al., 2022] that it often significantly improves AlphaFold2's confidence metrics. As displayed in Table 5.12, we came to a similar observation, as adding a small noise (with standard deviation 0.02 Å) slightly improves both pLDDT and pTM while slightly decreasing NSR. With bigger noise (standard deviation 0.2 Å), both tendencies are amplified. We reused the same level of noise as in the default version of ProteinMPNN.

Model	NSR	pLDDT (↑)	pTM (↑)
Native	100%	45.0	0.30
Effie (V2-multi)	44%	52.2	0.36
Effie (V2-multi-0.02)	44%	51.4	0.34
Effie (V2-multi-0.2)	39%	57.9	0.40
Effie (V3-multi)	53%	51.5	0.36
Effie (V3-multi-0.02)	50%	52.4	0.35
Effie (V3-multi-0.2)	44%	57.4	0.40

Table 5.12: NSR and forward folding metrics on different version of Effie, all trained on the multi-chain dataset. For comparison, native sequences have been folded as well. The decimal number in the version name indicates the amount of Gaussian noise in Å added to the coordinates during training (if the is none, no noise was used during training).

ProteinMPNN's authors offered a basic explanation for this preference: noising coordinates blurs out local details of the backbone geometry, hence the decrease of NSR, but it also helps the model focusing on overall topology, leading to sequences favoured by AlphaFold2. TERMinator's authors went deeper, pointing out some over-fitting by DL models [Li et al., 2023]. Two sequences only few mutations apart will often adopt the same structure, a few structural adjustments aside. Training on highly-specified crystal coordinates may result in over-fitting on specific residue

arrangements. We would like to add that this argument is supported by DL methods going way beyond 30% NSR, the biological threshold upon which 2 sequences can share the same fold. TERMinator's authors argue that noising coordinates reduces this bias, thus leading to better forward folding performances.

This observation is a further indicator that NSR is a limited metric. When it is high, additionnal metrics would be required to assess the utility of a learned score function. It also seems coherent with the fact that the designs by Effie (V3) are not considered better (even slightly poorer) by AlphaFold2 than the designs by other version of Effie, even though their NSR is much better. Effie (V3) may spot micro-details in the backbone, perhaps linked to crystallography, that are not significant in practice. Another explanation, that has not been discussed in previous work to our knowledge, comes from AlphaFold2 itself. The sequences produced by Effie (V3) are different from natural sequences as they are optimized. Thus, they might fall out of the distribution skill of AlphaFold2, which predicts pLDDT which does not correspond to reality. The hypothesis can only be validated by costly experimentation.

For practical designs, we favour forward folding over NSR and therefore we use noised Effie (multi) models. We wondered whether adding at inference the same level of noise as used during training would further improves the metrics, but it actually deteriorated all of them (see Table C.15 in Annex C). Further work could focus on perturbing the backbones in a more realistic fashion than Gaussian noise that is likely to break properties of a protein structure. It could be done with backrub or molecular modelling, but with a much higher computation cost.

5.2.4 Model Quality Assessment

Model quality assessment requires to assess the probability of a structure given a sequence $P(struct|seq)$. This task is common to assess physics-based energy scores, which predict the probability of both the sequence and the structure $P(seq, struct)$. However, Effie has not been trained on this objective but rather on predicting the probability of the sequence given the backbone $P(seq|struct)$. Yet, an unrealistic structure is likely to be ill-suited to the native sequence, explaining Effie's decent performance.

3DRobot Decoys

Since one of the inspirations for this work was the KORP potential, we compared the score function we learned with their knowledge-based potential. KORP did not target protein design, but it focused on model quality assessment (and loop modelling, but it is out of our scope). They identified or evaluated decoys, *i.e.,* artificial structures of various quality. These are standard tasks to assess physics-based energy functions as native structure or high-quality decoys should have a lower

energy. Even though Effie has not been trained for this task, we assessed whether its score is able to discriminate plausible structures.

One of their benchmark is 3DRobot [Deng et al., 2016], which contains 300 decoys for 200 proteins. The goal is to identify the target protein among the decoys. The target proteins are non-redundant (fewer then 20% identity) and span various domain classes (48α, 40β and 112α/β domains). Decoys were generated by an algorithm based on fragment assembly and they are diverse and are continuously distributed in RMSD space.

For a fair comparison, we removed redundancies between the training set and 3DR decoy set and we trained a new model on it. Since it was one of our first validation tasks, we worked with our first baseline, Effie V1, based on quaternion features. We followed the procedure from the KORP paper and we removed all sequences with more than 50% identity with one of the target protein (which is not very stringent). We used the software CD-hit-2d [Li and Godzik, 2006] in command line mode to cluster two sets of sequences with an identity threshold (exact parameters were `-c 0.5 -n 3 -d 0 -M 16000 -T 8 -s2 0`). It resulted in a dataset containing $17,202$ sequences.

Model	N_N	Z_N
KORP	193	3.43
Effie (V1)	**200**	**4.27**

Table 5.13: Comparison with KORP on the 3DRobot dataset.

For each target protein and its 300 decoys, we predicted the energy score of the native sequence over the backbone. The protein with the lowest score is predicted as the native protein. We used two metrics: N_N, the number of times the native proteins was identified (over 200) and Z_N, the standard deviation between the score of the native protein and the average score of the decoys. As displayed in Table 5.13, Effie identifies all the native proteins, and the gap between their score and the average decoy score is larger.

Rosetta Decoy Set

Another standard benchmark is the Rosetta decoy set [Park et al., 2016], composed of 133 protein structures each with thousand structure decoys each. The task is to sort these decoys based on their quality, measured by the TM score (a number between 0 and 1 indicating how well they align with the native structure).

Performances are measured in terms of Spearman's correlation between the score/energy of each decoy and its TM-score, and in terms of what is the TM-score of the best-ranked structure. We compared Effie (single) with the Rosetta energy

function ref2015 [Alford et al., 2017], DeepAccNet, a DL model for estimating the accuracy of protein structure models [Hiranuma et al., 2021], and an AlphaFold2's composite score developed in [Roney and Ovchinnikov, 2022]. All results are extracted from [Roney and Ovchinnikov, 2022]. There is only one version of Effie, the most recent one when this comparison was done.

Méthode	Spearman (median)	Spearman (mean)	TM best (median)	TM best (mean)
Effie (V2-single)	0.813	0.786	0.930	0.907
Rosetta	0.798	0.759	0.922	0.901
AF2-based	0.942	0.923	0.945	0.931
DeepAccNet	0.859	0.831	0.933	0.917

Table 5.14: Comparison on Rosetta Decoy set.

As shown in Table 5.14, Effie slightly outperforms Rosetta, which is remarkable because they do not have access to the same information: Effie is coarse-grained, while Rosetta's function is all-atom. Unsurprisingly, Effie is largely outperformed by DeepAccNet, which is dedicated to this task, and AlphaFold2.

Note that comparing DL methods that have not been trained on the same set is always risky, all the more so when the test set is not independent. Here, all the DL-based methods may have been trained on some of the structure of the Rosetta Decoy set. A fairer comparison would be the CASP14 set, which was not yet available at the time any of these models were trained [Roney and Ovchinnikov, 2022].

5.3 Applied Designs

After training and fine-tuning our model and assessing it *in silico*, we applied Effie to practical sequence design problems. Each of them has its own specificities requiring additional constraints or exact optimization. They illustrate some advantages of *a posteriori* conditioning and biasing.

First, following [Colom et al., 2022], we tried to enumerate variants of the SARS-CoV-2, with the additional idea of bounding the number of mutations, and recovered some of the existing variants. Second, we worked on a hexameric protein platform for synthetic biology using a negative multi-state design approach. We give some *in silico* and experimental evidence that our method is better suited to negative design. Third, we redesigned the core of the RNA polymerase using a constrained number of different amino acid types, showing how simpler chemistry could already have led to important proteins.

5.3.1 Enumeration of Potential SARS-Cov-2 Variants

To infect us, the spike protein of the SARS-Cov-2 binds to the human ACE2 receptor to form a complex and then proceed to penetrate human cells. The first structure of this complex was published in March 2020 [Walls et al., 2020]. The binding region is often denoted as the *Receptor Binding Domain* or RBD. Among the mutations observed in variants of SARS-Cov-2 [Philip et al., 2023], those in the RBD are of particular concern as they may increase the virus' ability to bind to human receptors and escape neutralizing antibodies. A representation of the RBD bound to ACE2 is given in Figure 5.5.

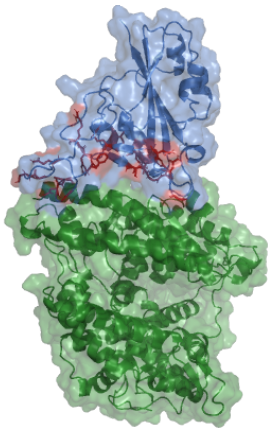


Figure 5.5: Structure of the region binding domain (RBD, in blue) bound to the human ACE2 receptor (in green). The 27 interface amino acid that can be redesigned are in red.

Following [Colom et al., 2022], we assume that the binding affinity between the

RBD and the ACE2 receptor increases with the difference in (free) energy between the bound and unbound states. We use the Effie score as an approximation of this energy.

$$-\Delta E = (E^{RBD} + E^{ACE2}) - E^{RBD+ACE2}$$

E^{RBD} is the energy of the RBD alone (unbound state) and $E^{RBD+ACE2}$ is the energy of the complex when the RB2 is bound to ACE2. To optimize its binding affinity, the virus would therefore benefit from a decreased stability of the unbound RBD, represented by $-E^{RBD}$. However, the RBD must maintain its structure, and to also avoid related entropic losses, its stability must remain sufficient. One can therefore assume that the viral evolution will tend to optimize ΔE while avoiding excessive destabilization of the RBD.

To model this situation, in their work, the authors only allow the 27 interface amino acids of RBD (called the RBM for Receptor Binding Motif) to mutate. The ACE2 sequence is known and fixed. The (free) energy is estimated by the full-atom Rosetta scoring function beta-nov16 [Pavlovicz et al., 2020]. Ideally, one would like to solve a difficult constrained multi-state design problem capturing the above objective for maximum affinity with an additional constraint bounding the energy of the RBD in its unbound form. Because of the extreme computational complexity of this problem (proved to be NP^{NP}-complete in [Vucinic et al., 2020]), the authors use a heuristic approach where they first enumerate low-energy bounded form sequences which are then filtered by removing those that give an excessively high energy to the unbound RBD.

However, the coarse-grain nature of Effie, which avoids all reasoning on the side-chain geometries, opens the door to a more elegant and possibly more tractable approach. We used Effie (V2.2-multi-0.02) to estimate the energies of sequences on the bound and unbound systems (PDB 6VXX processed as in [Colom et al., 2022]). Ideally, we could then search for sequences maximizing the binding affinity under the constraint that the energy of the RBD remains within e energy units of the wild type RBD energy E_{WT}:

$$\begin{aligned} &\min E^{RBD+ACE2} - E^{RBD} \\ &\text{s.t. } E^{RBD} < E_{WT} + e \end{aligned}$$

In this objective, the energy term E^{ACE2} is constant as the structure and sequence are assumed to be fixed. But this formulation requires to bound E^{RBD}, a full graphical model, which is currently not possible in existing Graphical Models optimization tools. We therefore consider a Lagrangian relaxation of the problem, which can then be written as:

$$\min E^{RBD+ACE2} - (1-\lambda)E^{RBD}$$

where the criteria to optimize is penalized by an amount proportional to E^{RBD}. The value of λ, our Lagrangian multiplier, is, however, unknown.

To determine a reasonable value for λ, we solved the above Lagrangian for various values of λ with toulbar2 and obtained a minimal energy E^*_λ for each λ. The best value for λ is one that defines an objective that is apparently optimized by the virus. We therefore decided to use the value of λ that gives the WT sequence a Lagrangian score that is closest to its minimum value, that is, the value of λ which minimizes:

$$(E_{WT}^{RBD+ACE2} - (1-\lambda)E_{WT}^{RBD}) - \min_s(E_s^{RBD+ACE2} - (1-\lambda)E_s^{RBD})$$

Using a simple grid search, we found the best value to be $\lambda = -0.5$ (see table 5.15).

λ	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0.0	0.1
ΔE	13.3	10.6	8.6	7.4	7.1	7.6	8.3	9.1	10.2	11.5	12.9

Table 5.15: Variation of ΔE wrt λ, computed for the wild-type sequence. Energy scores use the arbitrary units implicitly defined by Effie.

As the SARS-CoV2 virus evolves, it successively includes new mutations and must always remain functional, as captured by a sufficiently low value of the above Lagrangian (with $\lambda = -0.5$). To see if we could at least isolate the first pre-omicron variants of the SARS-CoV2, we enumerated all the sequences within 17 Effie units of the optimal score of the above Lagrangian that had up to 3 mutations. Among the 28 million such variant sequences, we used toulbar2 to enumerate $7,119,327$ sequences in less than 60 seconds on a recent Linux server, using only one thread. The list of enumerated variants includes all non-omicron important variants of SARS-Cov-2 (alpha, beta, delta, gamma, kappa, lambda, iota, mu). Indeed, Table 5.16 shows the distance to the optimum of the most important variants at this date and we can see that all non-omicron variants are within this 17 units threshold. Reaching the omicron variants would require tolerating 7 to 8 mutations in the RBM and allow for a score degradation above 18.80 which defines an extremely large sequence space. Indeed, the number of possible variants explode with the number of mutations, as displayed in Figure 5.6.

The score predicted by Effie is not very discriminatory since around 25% of sequences meet the scoring cutoff. This is also partly explained by the small number of allowed mutations. More sequences could be filtered out using probabilities based on mutation frequency between amino acids. These probabilities can be computed from SARS-Cov-2 DNA mutation rates [Sohpal, 2020] and codon (3 DNA letters) encoding each amino acid type. We leave this for future work.

5.3.2 The SpaceHex Project

Description of the Project

Bacterial microcompartments (BMC) are protein assemblies that spontaneously form in the cytoplasm of many bacteria. By encapsulating metabolic pathways,

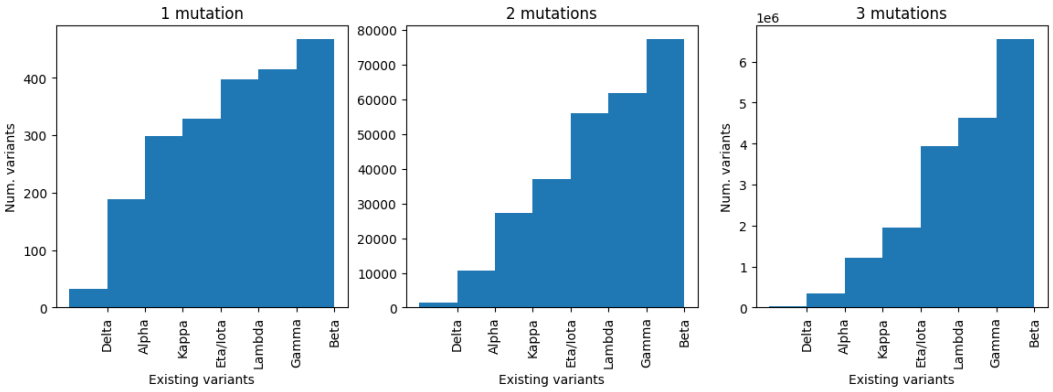


Figure 5.6: Number of variants with a maximum number of mutations (1, 2 or 3) with an energy score below the score of existing variant (x axis). The number of possible variants explodes as the number of mutations increases: with one mutation, less than 500 sequences meet the score cut-off, while with 3 mutations they are more than 7 millions.

they can optimize the yield of enzymatic reactions. Therefore, they are of great interest in synthetic biology to increase production yield.

BMCs are composed of hexamers (proteins with 6 chains), trimers and pentamers that form its facets, edges, and vertices, as illustrated in Figure 5.7. In this project, we focus on the hexamers. They are of interest in themselves as they can be used a platform for the spatial organisation of molecules in biotechnology applications. More precisely, we aim to design protein chains with specific interfaces (*i.e.*, with high interaction selectivity) such that they form heteromeric hexameric assemblies. We are interested in trimers of dimers ABABAB composed of two chains A and B (called heteromer AB in the following), and we want to avoid homo-hexamers AAAAAA or BBBBBB (called homomers AA and BB respectively). They are illustated in Figure 5.8.

From a Computational Protein Design perspective, this problem presents several characteristics. First, we aim to design symmetrical proteins. Thus, the CFN can be reduced to optimize a smaller equivalent problem. Second, since the succession of chains in each hexamer matters, multi-state negative approaches are required. The multi-state design allows us to consider several states simultaneously [Vucinic et al., 2020], and the negative design disfavours some of them. As discussed in the previous project, a coarse-grain function like Effie makes negative design much simpler as it avoids the optimization of the side-chains.

Mutant	No. mut.	Δ GMEC
20H (Beta V2) (B.1.351)	3	16.55
20I (Alpha V1) (B.1.1.7)	1	9.26
20J (Gamma V3) (P.1)	3	14.90
21A (Delta) (B.1.617.2)	0	7.13
21B (Kappa) (B.1.617.1)	1	11.12
21D (Eta) (B.1.525)	1	12.13
21F (Iota) (B.1.526)	1	12.13
21G (Lambda) (C.37)	0	7.13
21H (Mu) (B.1.621)	2	14.25
21K (Omicron) (BA.1)	7	18.90
21L (Omicron) (BA.2)	7	18.80
22A & 22B (Omicron) (BA.4&5)	7	20.52
22C (Omicron) (BA.2.12.1)	7	18.80
22D (Omicron) (BA.2.75)	7	20.05
22E (Omicron) (BQ.1)	8	20.52
22F (Omicron) (XBB)	8	23.70
23A (Omicron) (XBB.1.5)	8	24.24
23B (Omicron) (XBB.1.16)	8	24.24

Table 5.16: Variants of SARS-Cov-2, with their number of mutations (no. mut.) in the RBM and the distance to the optimal score computed on the Lagrangian problem with $\lambda = -0.5$.

Input Backbone and Designable Residues

The input backbone was constructed by Delphine Dessaux, post-doctoral fellow at TBI. She started from the chain A of an hexamer crystal structure forming a BMC (PDB id 5L38) [Mallette and Kimber, 2017]. Then a symmetrical hexamer conformation was built from this protein chain with Rosetta using symmetric facilities and minimized with Rosetta ref2015 score function. Each chain being composed of $n = 91$ residues, the input backbone is composed of $91 \times 6 = 546$ residues. A cartoon representation is displayed in Figure 5.9.

Only a partial redesign is done. For the design of specific interfaces, three different designable regions at the interface between two chains were defined. These regions are comprised of 21, 29 or 39 residues (they are named respectively small, interm. and large). All residue types are accepted.

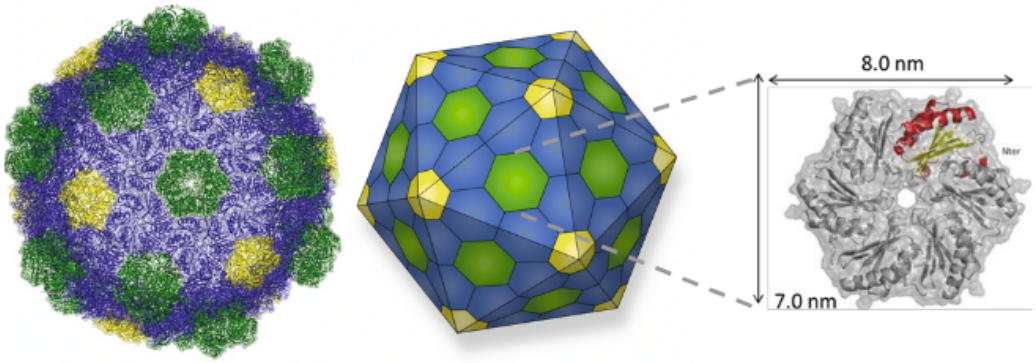


Figure 5.7: A bacterial micro-compartment (left), and its schematical representation (middle) with hexamers that form facets and edges. One hexamer is detailed on the right, with one of its 6 chains highlighted. Figure adapted from [Kirst and Kerfeld, 2019].

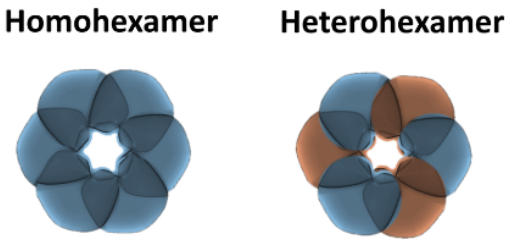


Figure 5.8: Simplified view of the homo-hexamer (6 identical chains, all in blue) and the hetero-hexamer (chains A in blue and chains B in red). Figure curtesy of D. Dessaux.

Reduction of the CFN

We predict our score function Effie as a CFN from the target backbone using the best version at this time, V2.1-multi-0.14.

Since the target protein is symmetrical, the CFN can be reduced to a smaller equivalent problem. The symmetry is of order 6 for the homomer AA (or BB) and of order 3 for the heteromer AB. Therefore, instead of problems with $6 \times n$ variables, they can be reduced to problems of respectively n and $2 \times n$ variables.

Figure 5.10 illustrates how the CFN is reduced in the case of the heteromer AB. The cost functions are stacked into a tensor of shape $6n \times 6n \times 20 \times 20$ representing the hexamer ABABAB, which will be turned into a $2n \times 2n \times 20 \times 20$ defining an equivalent problem on a dimer AB. To do so, the tensor is divided into 6×6 blocks, each representing the interactions between 2 chains. The possible interactions are AA, AB, BA and BB. Since only the upper triangular part is used to define the

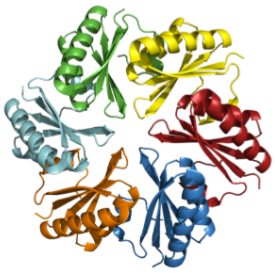


Figure 5.9: Cartoon view of the input backbone. The symmetry between the 6 chains is clearly visible.

costs, BA interactions are ignored (they are equivalent to AB interactions). All the AB blocks (in the upper part) are summed together.

For AA (or equivalently BB) interactions, some precautions must be taken, as illustrated in the bottom right par of the Figure. One block AA is composed of $n \times n$ sub-blocks of shape 20. Each sublock represent the interaction between 2 residues on a chain A. Two residues at the same position on 2 chains A must be the same, so only diagonal costs of diagonal sub-costs are kept. They represent unary costs. Moreover, sub-blocks on the lower part of block A are symmetrized and added to their upper block counterpart. Otherwise, these costs would be lost when defining the CFN.

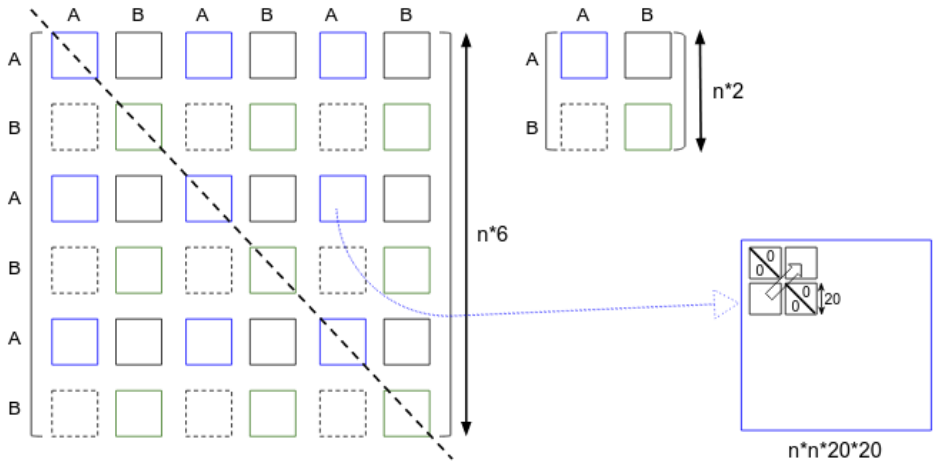


Figure 5.10: Reduction of the CFN of an hetero-hexamer ABABAB into an equivalent problem on a dimer AB. Each square corresponds to one block representing interactions between 2 chains (A and A in blue, B and B in green, and A and B in black).

Bi-objective Optimization

We produce 2 reduced CFNs per design region: one for the homomer AA (or BB) and one for the heteromer AB. Since we used coarse-grained representation, multi-state design boils down to adding the CFNs representing each state.

Our goal is for the heteromer to assemble, but not the homomer. Both CFNs were simultaneously optimized by Samuel Buchet, post-doctoral fellow at MIAT, using toulbar2 and the bi-criteria objective.

$$\begin{aligned} \min \quad & E_{AB}(seq_A, seq_B) \\ \max \quad & E_{AA}(seq_A) + E_{BB}(seq_B) \\ \text{s.t.} \quad & seq_A, seq_B \in D^d \end{aligned}$$

Where $E_{AA}(seq_A)$ is the energy of the homo-hexamer AA on the homomer backbone, D is the list of the 20 cananical amino acids and d is the number of residues to design. The non-designable residues are assigned to their native amino acid identity.

This bi-objective can be approximated with weights controlling the ratio between both objectives:

$$\min w_1 * E_{AB}(seq_A, seq_B) + w_2 * (E_{AA}(seq_A) + E_{BB}(seq_B))$$

with $w_1 \geq 0$ to favour the heteromer and $w_2 \leq 0$ to disfavour both homomers.

We varied the weights w_1 and w_2 to favour more or less one objective over the other using dichotomic method [Aneja and Nair, 1979]. For each pair of weights (w_1, w_2), one sequence was designed. In the first round of design described hereafter, we limited ourselves to the small and the interm. design regions because exact optimization on the large region becomes expensive computationally.

In Silico Validation

We designed a total of 60 sequences with Effie + toulbar2. We also designed sequences with ProteinMPNN [Dauparas et al., 2022] for comparison and to increase our chances of having one successful design. Indeed, ProteinMPNN has state-of-the-art *in silico* performances and it has been experimentally validated. If all the experiments reported on ProteinMPNN have been on single-state or multi-state design problems, authors mention that negative weights can be used for negative design and this option is implemented in the code they provide. Sequences were predicted by the default model, trained with a noise of 0.2Å standard deviation and sampled at $T = 0.1$. Weights ranging from -0.9 to -0.1 where used on the homomer (with a weight of 1 on the heteromer). For each weight, 10 sequences were predicted, resulting in a total of 90 sequences.

For each designed sequence, D. Dessaux achieved side-chain packing using toulbar2 plus Rosetta energy scoring functions. Indeed, all-atom functions are required

for this task. Then, she minimized the sequence using Rosetta ref2015 [Alford et al., 2017].

We first compared sequences designed by Effie+toulbar2 and by ProteinMPNN *in silico*. Since we want the heteromer to assemble, but not the homomer, our tests are based on the energy differences between both ($E_{AB}-E_{AA}$ and $E_{AB}-E_{BB}$). The more negative these two values are, the more likely the heteromer is to be favoured over the homomers. The first basic test was to verify that both these energies are negative for each sequence. Otherwise, it means the design did not disfavour the homomers. Since Effie is optimized, all its sequences have negative difference (Effie) scores. However, ProteinMPNN seems to struggle with negative design as 76/90 (84.4%) of its sequences have at least one positive difference (ProteinMPNN) score.

	Scoring		
Design	Effie	ProteinMPNN	betanov16
Effie	-195.2	-95.5	-1519.6
ProteinMPNN	35.4	-56.2	-1376.7

Table 5.17: The score $E_{AB} - E_{AA} + E_{AB} - E_{BB}$ assessed with 3 criteria (Effie, ProteinMPNN and betanov16) and averaged over all the sequences designed either with ProteinMPNN of Effie+toulbar2. When scoring with beta-nov16, sequences for which the heteromer AB was too destabilized (score > -1000) were excluded.

We then assessed the sequences of both models based on 3 criteria: Effie score, ProteinMPNN score and beta-nov16 score [Pavlovicz et al., 2020]. For some designs, beta-nov16 scored the target heteromer AB with very high energies, indicating it can not fold as desired. It happened for designs by both Effie and ProteinMPNN, and none of them were able to score these problematic designs as disfavourably as beta-nov16. Thus, when scoring with beta-nov16, we excluded designs distabilizing the heteromer AB too much. Since the wild type has an energy of -1314 beta-nov16 units we excluded sequences with a heteromer AB form scored lower than -1000 beta-nov16 units. This is an arbitrary cut-off that can be modified without impacting conclusions. Criteria cannot be directly compared as they do not share the same units. In Table 5.17, for each model we report the average score $E_{AB} - E_{AA} + E_{AB} - E_{BB}$ assessed with the three criteria.

Beta-nov16 is used as an independent score, and it shows a clear preference for Effie's sequences. Unsurprisingly, Effie's score favours Effie's sequence; it even considers that ProteinMPNN has failed to design negatively. More surprisingly, the ProteinMPNN score also prefers Effie's sequences. It means that the design problem defined by ProteinMPNN has lower energy solutions than those produced. ProteinMPNN was not able to find them by sampling but Effie could. We re-assessed

Effie and ProteinMPNN scores after minimization, and it showed the same trends. In short, on this negative design problem, ProteinMPNN is not good at optimizing its own criteria and it considers that Effie's sequences optimize it better. This is a direct consequence on designing sequences by sampling instead of optimizing.

Experimental Validation

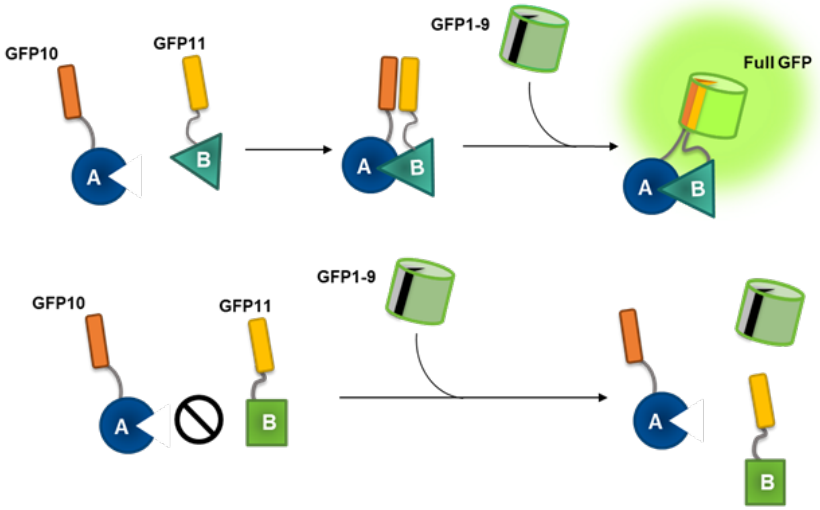


Figure 5.11: The tripartieGFP technology used for SpaceHex. Figure courtesy of Lucie Barthe, PhD student at TBI.

We further compared both approaches experimentally. The most promising sequences were selected based on the energy difference between the homomer and the heteromer. In total, 14 sequences were selected for Effie and 10 for ProteinMPNN. Experimental validation was carried out by Lucie Barthe and Luis Garcia-Alles from Toulouse Biotechnology Institute (TBI). To evaluate whether the hetero-hexamers could be formed, they assessed the quantity of AB interactions using the tripartite GFP technology [Cabantous et al., 2013], illustrated in Figure 5.11.

The Green Fluorescent Protein (GFP) is a protein with 11 β-strands. It is divided into 3 fragments: two β-strands (GFP10 and GFP11) and the rest of the protein (GFP9). GFP10 is linked to the monomer A and GFP11 to monomer B. If the two monomers interact, like in the hetero-hexamer ABABAB, they bring the two fragments closer together, which favours their binding to the GFP9 and the reconstruction of the full GFP. This reconstruction is visible through the fluorescence of the protein.

Therefore, a high level a fluorescence indicates more AB interactions. If only the homomers AA and BB form, there is no fluorescence. As shown in Figure 5.12,

sequences obtained with Effie show more fluorescence than sequences by ProteinMPNN, so they present more heteromers. Moreover, all the proteins designed by Effie saved one present a fluorescent signal above the negative control, while 4/10 designs by ProteinMPNN are below. For these proteins, no heteromers AB form, so the negative design apparently failed.

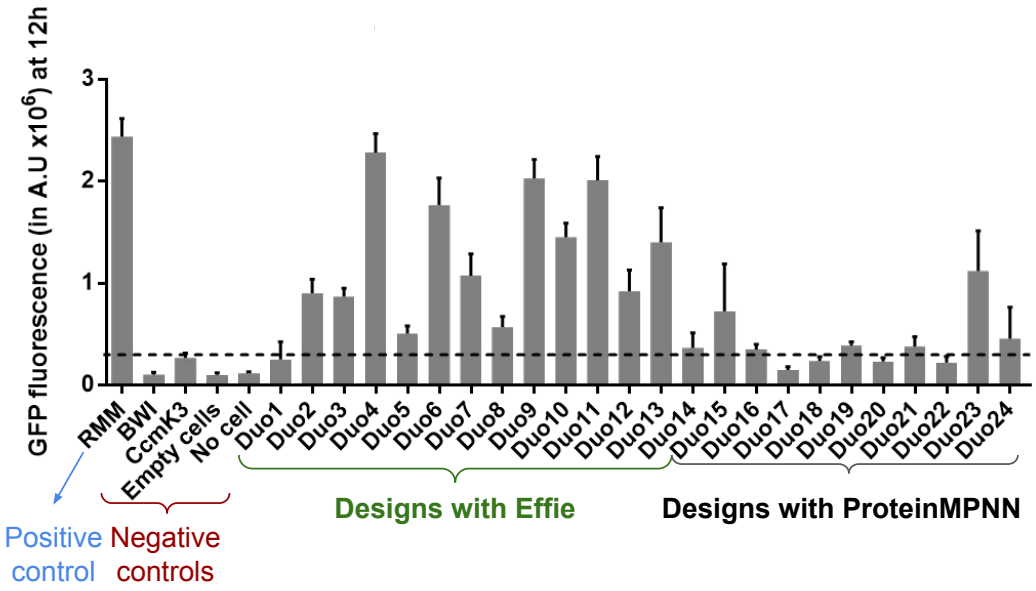


Figure 5.12: Experimental validation of designs obtained with Effie or ProteinMPNN. A high level of fluorescence indicates more AB interactions, suggesting more hetero-hexamers have been formed. The dotted line corresponds to the highest level of fluorescence of negative controls. A signal below this line cannot be interpreted as positive. Figure by Lucie Barthe.

5.3.3 Designing the Core Fold of RNA Polymerase

Complex protein structures are often assumed to have evolved from small and simple folds. One of these "prototype" folds is the double-ψ β-barrel (DPBB), which is present in several fundamental enzymes. In particular, the core domain of the RNA polymerase is composed of 2 DPBBs [Castillo et al., 1999].

Here, we take up previous work [Yagi et al., 2021] that aimed at showing that the crucial RNA polymerase enzyme could be synthesized with a simplified chemistry, involving only a small subset of all 20 natural amino acids (and associated codons). This would support the hypothesis that life could have started with such a simple RNA polymerase. In their paper, the DPBB fold was synthesized using only 7 types of amino acids. This was joint work between the teams in TBI, MIAT and the RIKEN Center for Biosystems Dynamics Research (Japan).

We tried to extend these results with fewer types of amino acids using Effie or ProteinMPNN. This setting is especially challenging as it aims for out-of-distribution generalization as, in our knowledge, no globular natural protein sequence is composed of so few types of amino acids.

Starting from one of their crystal structure (PDB 7DXZ), we computed the symmetry axis between the two monomers of the DPBB and we made the backbone precisely symmetric using AnAnaS [Pagès et al., 2018]. Backbones before and after pre-processing are displayed in Figure 5.13. Each monomer is composed of 42 amino acid residues.

(a) Crystal structure. **(b)** Predicted structure.

Figure 5.13: Crystal structure of the DPBB of RNA polymerase with 7 amino acid types and structure of one designed sequence with 6 types, predicted with AlphaFold.

We predicted the cost functions from the pre-processed backbone using Effie multi (trained with a noise of 0.02Å, 0.2Å or no noise). We constrained the number k of amino acid types using a N-value constraint [Pachet and Roy, 1999], encoded in toulbar2 with a generalized linear constraint [Montalbano et al., 2022] and additional Boolean variables, each indicating if a given type of amino acid is used. Without any constraint, the Effie-optimal sequence uses 12 different types of amino acids among the 20 available ones. We asked toulbar2 to find the optimal sequence while constraining the design with increasingly lower values of k. Figure 5.14 shows how Effie's score degrades as the constraint on the composition is tightened. From this curve, we decided to focus on designs using from 5 to 7 different amino acid types. We enumerated all solutions for $k \in \{5, 6, 7\}$ within 10 units of the optimum, and kept the 100 best solutions (in terms of predicted energy) for each value of k and for each Effie model (for $k = 5$, there were fewer than 100 solutions that were all kept).

We predicted the structure of $\backsim$ 450 sequences designed with Effie for $k = 6$ using AlphaFold (AF2) [Jumper et al., 2021]. The structure is predicted with LocalColabFold [Mirdita et al., 2022] without any template in single-sequence mode. In the best AF2 model (showing the lowest RMSD over all C_α w.r.t. the targeted

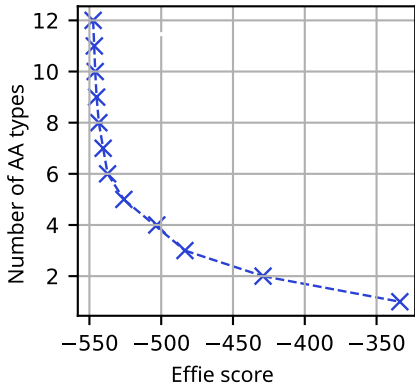


Figure 5.14: Optimum Effie's score (X-axis) as the constraint on the number of amino acid types used is tightened (Y-xis).

backbone structure), we observe that AF2 is able to almost perfectly recover the targeted structure ($RMSD = 1.31$Å), as shown in Figure 5.13b.

For each predicted structure, we collected the scores (pAE, pLDDT and pTM) and we computed the actual TM [Zhang and Skolnick, 2004] and Root Mean Square Deviation (RMSD), an alignment metric. Two types of RMSDs were computed:

- a local RMSD computed using the PyMol software: it rejects ill-aligned C_α, which results in an optimistic metric.
- a global RMSD over all C_α using a simple python-biotite script.

Model	Noise	$k = 7$	$k = 6$	$k = 5$
Effie v1	0	7	5	0
Effie v1	0.14	33	1	0
Effie v2	0	5	5	0
Effie v2	0.02	1	0	0
Effie v2	0.2	0	0	0
PMPNN1	Default	0	0	0
PMPNN2	Default	6	0	1
PMPNN3	Default	0	0	-

Table 5.18: Number of designed sequences with RMSD (global) < 3Å, $pLDDT > 60$ and $pAE < 10$, when designing with $k = 7$, $k = 6$ or $k = 5$ residues. PMPNN1 are the design made with ProteinMPNN on the first subset selected by Effie. All Effie models have been trained on multi-chain proteins.

Several sequences give interesting results, with RMSD (PyMol) < 2Å, $pLDDT > 60$ and $pAE < 10$. The pTM score is usually redundant with the other scores so we did not use it as an additional sequence filter. All the sequences satisfying those conditions have been gathered, and the associated most common subsets of amino acid types were selected. We kept 3 subsets for both $k = 6$ and $k = 7$ and 2 subsets for $k = 5$. For $k = 7$, the 3 subsets contain the amino acid DAGRV, with in addition either EL, TK or ET. This is in very good agreement with the subset that was experimentally tested in the original work [Yagi et al., 2021]: DAGRV-EK. For $k = 6$, all the subsets again contain DAGRV, with in addition L, T or E. For $k = 5$, the subset DAGRV is the most frequent and DTAGV is the second best.

Enforcing a N-value constraint on a sequence decoded auto-regressively, as done by pure-DL based models like ProteinMPNN, is not possible. Instead, we restricted ProteinMPNN [Dauparas et al., 2022] to design using only the subsets of amino acid types selected by Effie+toulbar2. This was achieved using the `omit_AAs` flag of ProteinMPNN, forbidding the 13, 14 or 15 types that were note not used in the corresponding Effie designs. We designed 100 ProteinMPNN sequences for each subset using a temperature $T = 0.01$.

In Table 5.18, we report the number of high-quality sequences obtained with each model. We define high-quality sequences based on global RMSD, pLDDT and pAE. We set the threshold to 3Å for RMSD to account for the more demanding global RMSD criteria. Still, the trends in the number of sequences for each model are the same. Selecting sequences based on the TM score alone, *i.e.,* the alignment between the target and the predicted structure, gives very similar results. We completed these results with violin plots to visualize all the designed sequences in Figure 5.15.

Concerning Effie sequences, the v1 seems to produce better quality sequences than the v2, both in terms of high-quality sequences and overall distributions. Moreover, the v2 no noise produces better sequences than noised models. It is the only task on which we observe an advantage of non-noised models over noised ones. This observation may be due to the input backbone really well fitting for the task. Overall, Effie predicts sequences of better quality than ProteinMPNN. Indeed, Effie sequences have a higher TM-score and there a more high-quality sequences. In particular, ProteinMPNN is able to produce high-quality sequences for only one subset of size $k = 7$, and none of size $k = 6$. However, it is the only model that produces a good-quality sequence for $k = 5$. This is surprising because the corresponding subset is also a subset of set tested for $k = 6$, where no high-quality sequence appeared. We assume that this is a direct consequence of the very constrained settings which forces ProteinMPNN to sample out-of-distribution sequences.

We have selected the most promising sequences with fewer than 7 amino acid types. They have been sent to our collaborators in Japan who are synthesizing and crystallizing them.

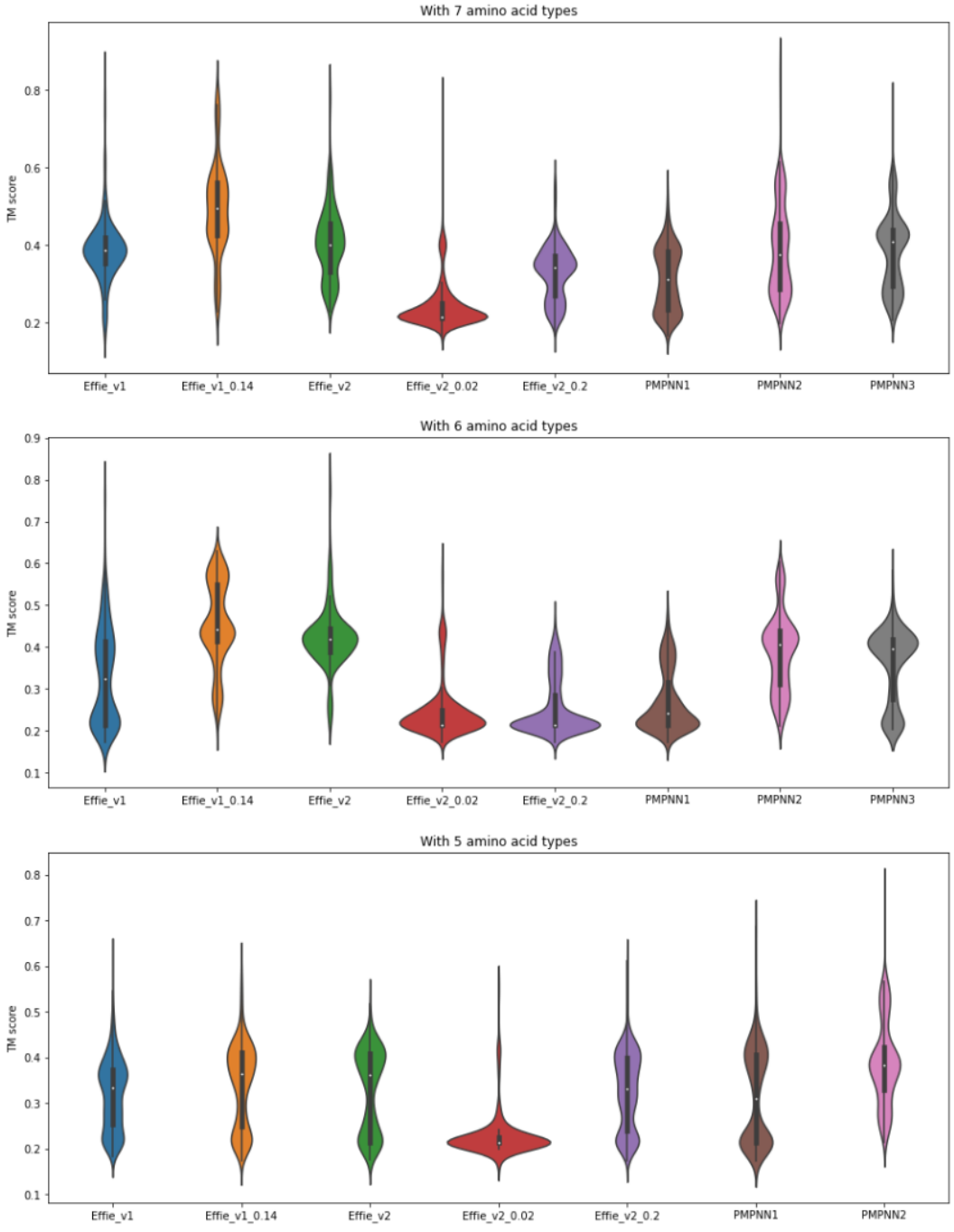


Figure 5.15: TM scores of all the designed sequences with Effie and ProteinMPNN models, allowing $k = 7$, 6 or 5 amino acid types.

Conclusion

In this chapter, we optimized Effie, our learned score function, to design proteins. The similarity of the designed and native sequences was compared to choose the hyper-parameters of the neural model. Once learned, Effie was assessed on various residue and protein-level tasks before being applied on practical design projects.

The *in silico* validation highlighted some limitations of the commonly-used metrics in design, particularly the NSR. Indeed, one of our variant architectures offers a large boost in NSR which is not reflected on other tests, both mutation effect prediction and forward folding. In those conditions, the comparison between neural architectures solely based on NSR seem uninformative in terms of design utility. We did not work with perplexity as it is intractable in our case, but it is likely to suffer from the same shortcoming as NSR.

Design utility also comes from the ability to fulfill design objectives. We argued throughout this work that optimization can offer practical advantages that are out-of-reach of other DL models that design the sequence either auto-regressively or by direct mapping. Our applied projects present specific problems that are difficult for sampling-based methods, like the negative design for SpaceHex, and problems that can be only be tackled through optimization, like enumerating of sub-optimal solutions and limitations of the number of mutations to predict covid variants and restricting of the number of amino acid type to design a DPBB. We offer evidence of the short-comings of pure DL models by using ProteinMPNN, an approach thought for practical design [Dauparas et al., 2022] and experimentally validated [Yeh et al., 2023], on the same tasks.

In practice, predicting a sequence folding on a target structure is only a part of the design pipeline. The designed sequences are selected *in silico* and evaluated experimentally. One of the post-processing step is to adapt the structure to the sequence through relaxation. Since it is usually done using Rosetta energy functions, we wondered whether Effie could be used as well. We did some preliminary work to update the structure coordinates to the designed sequence based on an adversarial attack method [Goodfellow et al., 2014], constrained to respect characteristics of a protein structure. This work is describe in Annex D, but the results we obtained so far seem encouraging but non-conclusive.

Other future work can be considered. Practical designs often involve ligands, which are not composed of residue. Therefore, the interaction between a protein and a ligand cannot be predicted by Effie (or any other DL methods, which are all coarse-grain). Yet, Effie could be combined with an all-atom potential [Alford et al., 2017; Pavlovicz et al., 2020] that will be in charge of modelling protein-ligand interaction, while Effie would be used to represent intra-protein interactions.

Concerning the neural net itself, it is likely to benefit from data augmentation

using predicted structures, as done in [Hsu et al., 2022]. Moreover, training could be revisited. If a first stage under the E-NPLL seems necessary, a second stage under the Hinge loss could improve the quality of Effie at it would consider the protein as a whole. The Hinge loss on the complete protein would obviously be intractable, but alternative schemes can be used. We briefly tried to design only a small sphere around a randomly selected residue to limit the optimization problem to about 20 residues. It was not directly conclusive, but since we improved the model and we consider more metrics than the sole NSR, it will be interesting to further explore this idea.

Bibliography

Alford, R. F., Leaver-Fay, A., Jeliazkov, J. R., O'Meara, M. J., DiMaio, F. P., Park, H., Shapovalov, M. V., Renfrew, P. D., Mulligan, V. K., Kappel, K., et al. (2017). The rosetta all-atom energy function for macromolecular modeling and design. *Journal of chemical theory and computation*, 13(6):3031–3048.

Anand, N., Eguchi, R., Mathews, I. I., Perez, C. P., Derry, A., Altman, R. B., and Huang, P.-S. (2022). Protein sequence design with a learned potential. *Nature communications*, 13(1):746.

Aneja, Y. P. and Nair, K. P. K. (1979). Bicriteria transportation problem. *Management Science*, 25(1):73–78.

Bennett, N. R., Coventry, B., Goreshnik, I., Huang, B., Allen, A., Vafeados, D., Peng, Y. P., Dauparas, J., Baek, M., Stewart, L., et al. (2023). Improving de novo protein binder design with deep learning. *Nature Communications*, 14(1):2625.

Bernhofer, M. and Rost, B. (2022). Tmbed: transmembrane proteins predicted through language model embeddings. *BMC bioinformatics*, 23(1):326.

Cabantous, S., Nguyen, H. B., Pedelacq, J.-D., Koraïchi, F., Chaudhary, A., Ganguly, K., Lockard, M. A., Favre, G., Terwilliger, T. C., and Waldo, G. S. (2013). A new protein-protein interaction sensor based on tripartite split-gfp association. *Scientific reports*, 3(1):2854.

Castillo, R. M., Mizuguchi, K., Dhanaraj, V., Albert, A., Blundell, T. L., and Murzin, A. G. (1999). A six-stranded double-psi β barrel is shared by several protein superfamilies. *Structure*, 7(2):227–236.

Colom, M. S., Vucinic, J., Adolf-Bryfogle, J., Bowman, J. W., Verel, S., Moczygemba, I., Schiex, T., Simoncini, D., and Bahl, C. D. (2022). Deep evolutionary forecasting identifies highly-mutated sars-cov-2 variants via functional sequence-landscape enumeration. *Research Square*, pages rs–3.

Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Courbet, A., de Haas, R. J., Bethel, N., Leung, P. J. Y., Huddy, T. F., Pellock, S., Tischer, D., Chan, F., Koepnick, B., Nguyen, H., Kang, A., Sankaran, B., Bera, A. K., King, N. P., and Baker, D. (2022). Robust deep learning-based protein sequence design using proteinMPNN. *Science*, 378(6615):49–56.

Deng, H., Jia, Y., and Zhang, Y. (2016). 3drobot: automated generation of diverse and well-packed protein structure decoys. *Bioinformatics*, 32(3):378–387.

Durante, V., Katsirelos, G., and Schiex, T. (2022). Efficient low rank convex bounds for pairwise discrete Graphical Models. In *Thirty-ninth International Conference on Machine Learning*.

Gao, Z., Tan, C., and Li, S. Z. (2023). Pifold: Toward effective and efficient protein inverse folding. In *The Eleventh International Conference on Learning Representations*.

Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Goverde, C., Wolf, B., Khakzad, H., Rosset, S., and Correia, B. E. (2023). De novo protein design by inversion of the alphafold structure prediction network. *Protein science : a publication of the Protein Society*, page e4653.

Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919.

Hiranuma, N., Park, H., Baek, M., Anishchenko, I., Dauparas, J., and Baker, D. (2021). Improved protein structure refinement guided by deep learning based accuracy estimation. *Nature communications*, 12(1):1340.

Hsu, C., Verkuil, R., Liu, J., Lin, Z., Hie, B., Sercu, T., Lerer, A., and Rives, A. (2022). Learning inverse folding from millions of predicted structures. In *International Conference on Machine Learning*, pages 8946–8970. PMLR.

Hurley, B., O'sullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., and Givry, S. d. (2016). Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, 21:413–434.

Ingraham, J., Garg, V. K., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Jankauskaitė, J., Jiménez-García, B., Dapkūnas, J., Fernández-Recio, J., and Moal, I. H. (2019). Skempi 2.0: an updated benchmark of changes in protein–protein binding energy, kinetics and thermodynamics upon mutation. *Bioinformatics*, 35(3):462–469.

Jing, B., Eismann, S., Suriana, P., Townshend, R. J. L., and Dror, R. (2021). Learning from protein structure with geometric vector perceptrons.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations - ICLR, San Diego - Conference Track Proceedings*.

Kirst, H. and Kerfeld, C. A. (2019). Bacterial microcompartments: catalysis-enhancing metabolic modules for next generation metabolic and biomedical engineering. *BMC biology*, 17:1–11.

Li, A. J., Lu, M., Desta, I., Sundar, V., Grigoryan, G., and Keating, A. E. (2023). Neural network-derived potts models for structure-based protein design using backbone atomic coordinates and tertiary motifs. *Protein Science*, 32(2):e4554.

Li, W. and Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., et al. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130.

Lu, H., Diaz, D. J., Czarnecki, N. J., Zhu, C., Kim, W., Shroff, R., Acosta, D. J., Alexander, B. R., Cole, H. O., Zhang, Y., et al. (2022). Machine learning-aided engineering of hydrolases for pet depolymerization. *Nature*, 604(7907):662–667.

Mallette, E. and Kimber, M. S. (2017). A complete structural inventory of the mycobacterial microcompartment shell proteins constrains models of global architecture and transport. *Journal of Biological Chemistry*, 292(4):1197–1210.

Mirdita, M., Schütze, K., Moriwaki, Y., Heo, L., Ovchinnikov, S., and Steinegger, M. (2022). Colabfold: making protein folding accessible to all. *Nature methods*, 19(6):679–682.

Montalbano, P., de Givry, S., and Katsirelos, G. (2022). Multiple-choice knapsack constraint in graphical models. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 282–299. Springer.

Pachet, F. and Roy, P. (1999). Automatic generation of music programs. In *International Conference on Principles and Practice of Constraint Programming*, pages 331–345. Springer.

Pagès, G., Kinzina, E., and Grudinin, S. (2018). Analytical symmetry detection in protein assemblies. i. cyclic symmetries. *Journal of Structural Biology*, 203(2):142–148.

Park, H., Bradley, P., Greisen, P., Liu, Y., Mulligan, V. K., Kim, D. E., Baker, D., and DiMaio, F. (2016). Simultaneous optimization of biomolecular energy functions on features from small molecules and macromolecules. *Journal of Chemical Theory and Computation*, 12(12):6201–6212.

Pavlovicz, R. E., Park, H., and DiMaio, F. (2020). Efficient consideration of coordinated water molecules improves computational protein-protein and protein-ligand docking discrimination. *PLoS Computational Biology*, 16(9):e1008103.

Philip, A. M., Ahmed, W. S., and Biswas, K. H. (2023). Reversal of the unique q493r mutation increases the affinity of omicron s1-rbd for ace2. *Computational and Structural Biotechnology Journal*, 21:1966–1977.

Pierce, N. A. and Winfree, E. (2002). Protein Design is NP-hard. *Protein Engineering, Design and Selection*, 15(10):779–782.

Rocklin, G. J., Chidyausiku, T. M., Goreshnik, I., Ford, A., Houliston, S., Lemak, A., Carter, L., Ravichandran, R., Mulligan, V. K., Chevalier, A., et al. (2017). Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science*, 357(6347):168–175.

Roney, J. P. and Ovchinnikov, S. (2022). State-of-the-art estimation of protein model accuracy using alphafold. *Physical Review Letters*, 129(23):238101.

Ruffini, M., Vucinic, J., de Givry, S., Katsirelos, G., Barbe, S., and Schiex, T. (2019). Guaranteed diversity & quality for the weighted csp. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pages 18–25. IEEE.

Simoncini, D., Allouche, D., De Givry, S., Delmas, C., Barbe, S., and Schiex, T. (2015). Guaranteed discrete energy optimization on large protein design problems. *Journal of chemical theory and computation*, 11(12):5980–5989.

Sohpal, V. K. (2020). Computational analysis of sars-cov-2, sars-cov, and mers-cov genome using mega. *Genomics & Informatics*, 18(3).

Townshend, R. J., Vögele, M., Suriana, P., Derry, A., Powers, A., Laloudakis, Y., Balachandar, S., Jing, B., Anderson, B., Eismann, S., et al. (2020). Atom3d: Tasks on molecules in three dimensions. *arXiv preprint arXiv:2012.04035*.

Vucinic, J., Simoncini, D., Ruffini, M., Barbe, S., and Schiex, T. (2020). Positive multistate protein design. *Bioinformatics*, 36(1):122–130.

Walls, A. C., Park, Y.-J., Tortorici, M. A., Wall, A., McGuire, A. T., and Veesler, D. (2020). Structure, function, and antigenicity of the sars-cov-2 spike glycoprotein. *Cell*, 181(2):281–292.

Yagi, S., Padhi, A. K., Vucinic, J., Barbe, S., Schiex, T., Nakagawa, R., Simoncini, D., Zhang, K. Y., and Tagami, S. (2021). Seven amino acid types suffice to create the core fold of rna polymerase. *Journal of the American Chemical Society*, 143(39):15998–16006.

Yang, J., Anishchenko, I., Park, H., Peng, Z., Ovchinnikov, S., and Baker, D. (2020). Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences*, 117(3):1496–1503.

Yeh, A. H.-W., Norn, C., Kipnis, Y., Tischer, D., Pellock, S. J., Evans, D., Ma, P., Lee, G. R., Zhang, J. Z., Anishchenko, I., et al. (2023). De novo design of luciferases using deep learning. *Nature*, 614(7949):774–780.

Zhang, Y. and Skolnick, J. (2004). Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, 57(4):702–710.

Conclusion

This thesis presented a new hybrid approach for Computational Protein Design (CPD) combining Deep Learning and Automated Reasoning. After introducing all the required notions, we defined the problem of structure-based protein design. When reviewing existing methods in Chapter 2, we distinguished traditional energy-based methods from more recent pure DL-based methods. We noted that the latter offers better performances, both *in silico* and experimentally, but the former has practical advantages as additional design objectives and constraints can be represented and enforced. In this work, we aimed to bring the best of both worlds into a hybrid method. More precisely, we aimed to take advantage of existing protein structures by learning a scoring function that was optimized to design new proteins.

This formulation of CPD is a challenging unsupervised problem as the score function must be learned from examples of good solutions alone. In Chapter 3, we placed our problem within the framework of Decision-Focused Learning. Since none of the existing approaches offered both scalability to large instances and exact solving at inference, we introduced a new loss, the Emmental-PLL, to learn a Graphical Model that is solved only at inference. On the toy problem of learning how to play Sudoku, and some variants, our hybrid pipeline presented state-of-the-art results. This problem was the occasion to illustrate some advantages in combining learning and reasoning, including interpretability, data efficiency, and the possibility to add *a posteriori* knowledge or constraints. All these properties are desirable in CPD too.

Protein structures, being non-Euclidean data, set new learning challenges. A consensus about the most suited representation has yet to be reached. In Chapter 4, we described the data representation and neural architecture we used to learn Effie, our scoring function in the form of a Graphical Model. Effie was optimized in Chapter 5 and extensively assessed *in silico*. It outperformed energy-based methods and was competitive with pure DL-based approaches for designing proteins. Moreover, Effie can tackle other related tasks for which it had not been trained, suggesting that some physics-chemical concepts have been learned. Finally, we demonstrated the advantages of our hybrid approach over pure DL methods on applied design projects, one of them being experimentally validated.

Perspectives

Our work could be extended in two directions: Computational Protein Design and hybrid AI.

From a hybrid-AI perspective, we could extend the Emmental-PLL to other applications. Since it is agnostic to the neural net, any problems that can be represented as a CFN/MRF can potentially be tackled. In practice, memory consumption will limit the Graphical Model to include terms between at most 3 variables (on small problems). To alleviate this issue, we could use latent variables to encode more complex interactions with pairwise terms. Moreover, the Hinge loss has the ability to consider a discrete problem as a whole and therefore not learn redundant constraints. When this property is desired, we could further explore the convergence of training under the Hinge loss but with an approximate solver for more tractability. Such training may also benefit protein design.

In CPD, the confidence we can have in Effie's quality will build over the course of practical designs it will be applied to. Therefore, our main perspective is to apply it to as many projects as possible. Particularly challenging tasks are the one including interactions between several molecules, proteins or not. They are also highly interesting as many functions occur through interactions. Moreover, we now have several versions of Effie with variable performances depending on the task. Yet, we do not completely understand which version is the best for a given task, so further exploration is needed. Results obtained using various versions of Effie also opened questions about the *in silico* assessment of designed methods and designed sequences. It is necessary, as large-scale experimental validation is not realistic, but the metrics used can disagree. A more unified and robust *in silico* assessment would be useful. Suggesting one will require to have a better understanding of the limitations of AlphaFold2, which is often trusted as ground truth even though its distribution of expertise is not well known, especially in single-sequence mode with no template.

Designing a protein goes beyond the prediction of a sequence for an input backbone. A longer-term objective would be to automatize the whole design process. For instance, diffusion models could be used to generate the input backbone and the predicted backbone could be minimized through Effie's relaxation. Then, the solver toulbar2 could be asked to predict more sequences which will be automatically filtered out based on forward folding. In the case of several rounds of experimental validation, active learning could be used to incorporate early results to find more suitable candidate sequences in subsequent rounds. Finally, our formulation of CPD itself could be challenged. Many restricting hypothesis are used so that the problem can be tackled. In particular, the backbone is considered rigid while most functions involve protein flexibility. This can be only partially alleviated by considering multiple input backbones.

Appendix A

Gradients of the NPLL

The negative pseudo log-likelihood of the dataset S is the sum of the negative log-probability of each $(\mathbf{y}^l)$:

$$\begin{aligned} NPLL(S) &= \sum_{l=1}^{m} NPLL(\mathbf{y}^l) \\ &= -\sum_{l=1}^{m} \left[\sum_{Y_i \in \mathbf{Y}} \log P^{\mathcal{M}}(y_i^l | \mathbf{y}_{-i}^l) \right] \end{aligned}$$

where

$$P^{\mathcal{M}}(y_i^l | \mathbf{y}_{-i}^l) = \frac{\exp(-\sum_{j \neq i} \mathcal{M}[i,j](y_i^l, y_j^l))}{\sum_{v_i \in D^i} \exp(-\sum_{j \neq i} \mathcal{M}[i,j](v_i, y_j^l))}$$

The conditional probability above is obtained using the normalizing constant $Z^{\mathcal{M}}(\mathbf{y}_{-i}^l)$ in the denominator:

$$Z^{\mathcal{M}}(\mathbf{y}_{-i}^l) = \sum_{v_i \in D^i} \exp(-\sum_{j \neq i} \mathcal{M}[i,j](v_i, y_j^l))$$

computed over all possible values v_i of Y_i. This corresponds to the application of a softmax on the logits $-\sum_{j \neq i} \mathcal{M}[i,j](v_i, y_j^l)$.

Minimizing the NPLL means maximizing the probability above, therefore making $-\mathcal{M}[i,j](\cdot, y_j^l)$ higher on the observed value y_i^l (used in the numerator) than on the other values $v_i \neq y_i^l$ or equivalently, the cost $\mathcal{M}[i,j](\cdot, y_j^l)$ lower on y_i^l than on other values: the NPLL is a contrastive loss that seeks to create a margin between

the values that are observed in the sample S and the other values of the variable, for every variable and every sample.

Focusing on one sample $\mathbf{y} \in S$, we expand and get:

$$NPLL(\mathbf{y}) = - \sum_{Y_i \in \mathbf{Y}} \left[\left(- \sum_{j \neq i} \mathcal{M}[i,j](y_i, y_j) \right) - \log Z^{\mathcal{M}}(\mathbf{y}_{-i}) \right] \quad \text{(A.1)}$$

The NPLL is a sum over all variables $Y_i \in \mathbf{Y}$ and we consider the contribution of a given variable Y_i. To compute the gradients of the corresponding term of the NPLL, we first compute the partial derivative of the logarithm of the normalizing constant $Z^{\mathcal{M}}(\mathbf{y}_{-i})$ (i fixed) w.r.t. $\mathcal{M})[i,j](v_i, y_j)$ (for arbitrary $j \neq i$ and $v_i \in D^i$, other costs do not appear in $Z^{\mathcal{M}}(\mathbf{y}_{-i})$ and the corresponding partial derivative is 0).

$$\begin{aligned} \frac{\partial \log Z^{\mathcal{M}}(\mathbf{y}_{-i})}{\partial \mathcal{M}[i,j](v_i, y_j)} &= \frac{-\exp(-\sum_{k \neq i} \mathcal{M}[i,k](v_i, y_k))}{Z^{\mathcal{M}}(\mathbf{y}_{-i})} \\ &= -P^{\mathcal{M}}(v_i | \mathbf{y}_{-i}) \end{aligned}$$

For any Y_i, the partial derivative of the first term in equation A.1 w.r.t. $\mathcal{M}[i,j](v_i, v_j)$ is $-\mathbb{1}(v_i = y_i, v_j = y_j)$.

Overall, given that $\mathcal{M}[i,j](v_i, v_j)$ and $\mathcal{M}[j,i](v_j, v_i)$ are the same, the contribution of sample $(\omega, \mathbf{y})$ to $\frac{\partial NPLL}{\partial \mathcal{M}[i,j](v_i, v_j)}$ will reduce to the non-zero contributions of variables Y_i and Y_j:

$$\begin{aligned} \frac{\partial NPLL}{\partial \mathcal{M}[i,j](v_i, v_j)} = &[\mathbb{1}(y_i = v_i, y_j = v_j) - P^{N(\omega)}(v_i | \mathbf{y}_{-i}) \mathbb{1}(y_j = v_j)] \\ &+ [\mathbb{1}(y_i = v_i, y_j = v_j) - P^{N(\omega)}(v_j | \mathbf{y}_{-j}) \mathbb{1}(y_i = v_i)] \end{aligned}$$

Appendix B

Parameters of LR-BCD

Parameter Selection

LR-BCD has 3 parameters to be chosen: the number of iterations, the rank and the number of roundings. We experimentally chose each of them by fixing 2 parameters and make the last vary. We randomly selected 17 proteins with less than 100 amino acids chosen randomly from the single-chain test set. For each protein and each number of rounding to test, we run LR-BCD 10 times and plot the span in energy observed on the 10 designed sequences. Results are in Figure B.1.

NSR Computation

LR-BCD produces as many solutions as the number of roundings (denoted nR). To compute the NSR, we compared 3 methods:

- Mean NSR on the nR predictions
- NSR of the solution with the lowest predicted energy (denoted E_0)
- Mean NSR, weighted by the energy softmax $\frac{e^{-(E_i-E_0)/m}}{\sum_{j=1}^{nR} e^{-(E_j-E_0)/m}}$, with n the number of residues

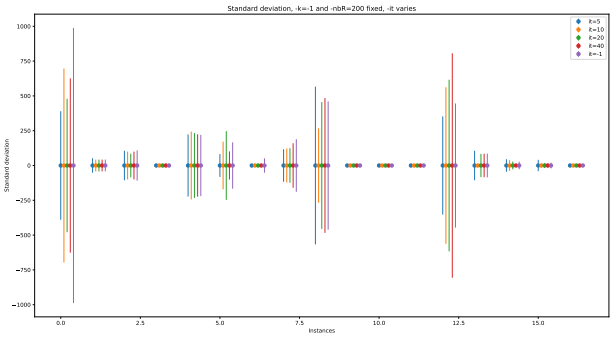
Comparison was done on the single-chain test set, with Effie V1-single. For each protein, the NSR was computed with one of the methods (named respectively mean, best and softmax), then we took either the mean or the median over the whole test set. In both cases, selecting the sequence with the lowest energy gave the best results (see Table B.1).

NSR	Mean	Best	Softmax
Mean	33.4%	**34.1%**	33.5%
Median	34.4%	**35.3%**	34.5%

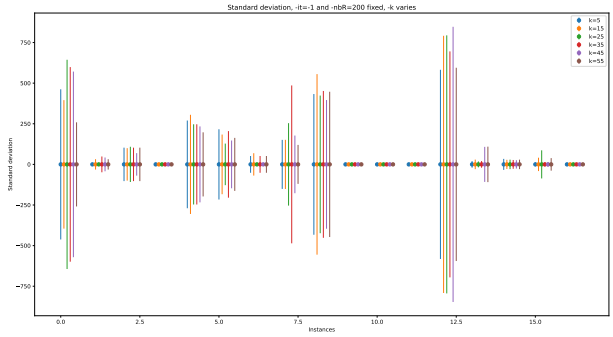
Table B.1: Comparison on the 3 methods to compute the NSR over one prediction of LR-BCD.

Comparison with Toulbar2 no Backtrack

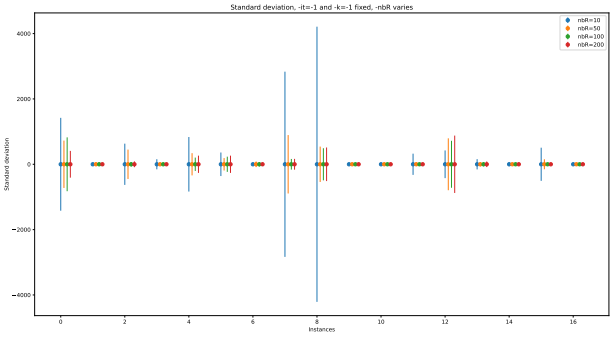
We compared LR-BCD and toulbar2 with no backtrack on the single-chain test set. Both give similar results in terms of NSR, but LR-BCD inference time is almost independent of the size of the instance, while toulbar2 with no backtrack presents a linear trend between both.



(a) Impact of the number of iteration.



(b) Impact of the rank.



(c) Impact of the number of roundings.

Figure B.1: Impact of each parameter of LR-BCD on the energy span between solutions of 10 independent runs. Experiments made with Valentin Durante, PhD at MIAT and designer of the LR-BCD algorithm.

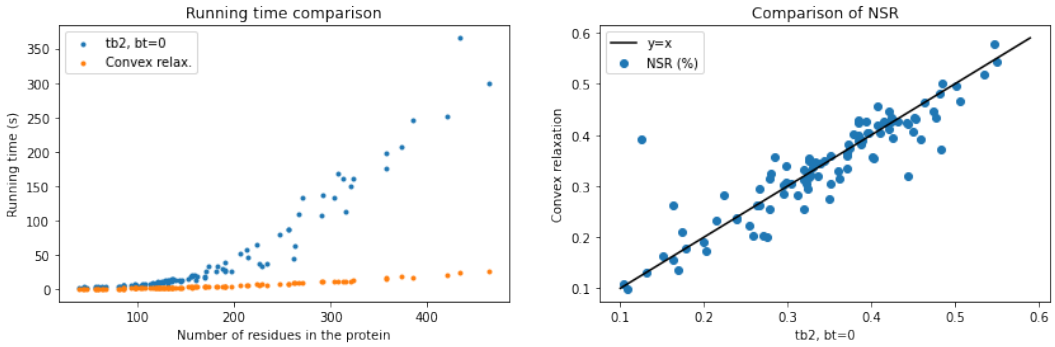


Figure B.2: Comparison between LR-BCD and toulbar2 with no backtrack. Left: inference time on each instance. Right: NSR on each instance.

Appendix C

Effie's Hyperparameters

Each hyper-parameter was chosen by fixing all the others and making it vary. If the resulting model resulted in improved NSR or a simpler model, it hyper-parameter was selected. In our experiments, choices were made on the per-residue accuracy, which is a good indicator of the final NSR.

For each set of experiments, a table is given. The first row is the current baseline with which comparisons are made. All results are given on validation sets, and by default we worked on the single-chain dataset.

C.1 First Baseline: Quaternion Features

Our first baseline used quaternion features. This section describes how its hyper-parameters were chosen to come up with Effie V1.

Optimization and Architecture Parameters

Our first baseline has the following parameters:

- Relative-position features
  - Translation unit vector and quaternion
  - Positional embedding
  - Contact numbers, dihedral angles
- Architecture:
  - gMLP: seq_len= 45, depth= 12, ff_mult = 4, out_dim = 128
  - resMLP: n_block = 15, block_size = 2, hidden = 128

- Optimization:
  - initial LR = 10^{-4}, cut to 10^{-5} when validation loss increases, $wd = 10^{-4}$, Adam optimizer
  - No regularization over the predicted GM (we tried L1 and L2)
  - No distance cut-off between pairs of variables (indicated by thresh)

Difference to the baseline	Acc.	NSR	Conclusion
-	41.0%	18.1%	**Baseline**
LR = 10^{-3}	40.1%	-	initial LR 1e-4 + decrease
weight_decay = 0.001	**41.2%**	-	keep + regularization
gMLP out_dim=32	41.0%	-	gMLP out_dim = 32
ff_mult=2	39.5%	-	keep ff_mult=4
embed_dim=64	40.3%	-	keep embed_dim=64
thresh = 10	40.4%	33.6 %	thresh=10
ResMLP hidden size=64	38.7%	-	keep width ResNet to 128
depth gMLP=15, nblocks = 20, gMLP out_dim=32	**41.3%**	-	Increase the capacity of both nets
Multi-task, alpha = 0.5	41.0%	33.2%	No multi-task
Multi-task, alpha = 0.75	41.1%	33.2%	No multi-task
Regularization: 0.0001 L1	**41.1**%	30.2%	Keep L1
Regularization: 0.0001 L2	40.9\%	14.7%	No L2

Table C.1: Variation of different parameters of the first baseline (in terms of optimization, architecture and regularization) and the resulting performances.

Unary Terms and Environment Feature Extraction

The baseline only predicts binary terms. We added the prediction of unary terms and we tried multi-task learning by feeding them to a cross-entropy loss. The total loss is $\alpha PLL + (1 - \alpha)CE$. Results (in Table C.2) show it does not lead to better NSR than the baseline with a L1-regularization or a distance cut-off.

Concerning the architecture we used to extract the information about the environment of a single residue, we compared gMLP to GPT2 architecture. We also compared 2 ways of aggregating the output into a single feature vector: either by averaging (mean), or by taking the line representing the target residue (central). Results in Table C.3 focuses on prediction using unary terms only (thus training was done with the sole cross-entropy). The gMLP architecture lead to better NSR than GPT2, and averaging their output representation is clearly better.

Learning task	Accuracy	NSR	Conclusion
Single-task ($\alpha = 0$)	41.0%	18.1%	**Baseline**
Single-task ($\alpha = 0$)	40.4%	33.6%	With thresh=10
Multi-task, $\alpha = 0.5$	41.0%	33.2%	No multi-task
Multi-task, $\alpha = 0.75$	41.1%	33.2%	No multi-task

Table C.2: Comparison of models trained to predict both unary and binary terms or unary terms only.

Envt ft vector	Archi	Accuracy	# Params.
Central	GPT	29.8%	37.9M
Mean	GPT	30.6%	37.9M
Central	gMLP	22.8%	10.9M
Mean	gMLP	**32.6%**	10.9M

Table C.3: Comparison of architecture to extract the environment feature vector.

New Baseline

Based on the conclusion drawn from each training in Table C.1, the capacity of both nets is increased. After tuning other parameters, we realized the capacity was larger than necessary. Predicting both unary and binary terms, and applying multi-task learning do not increase performance, so we focus on binary terms in the following.

Using a distance cut-off between pairs of residues increase speed up to 3-fold and it increases NSR (even though it slightly reduces accuracy), so we keep it. In terms of regularization on the predicted GM, the L1 improves over the L2, and it increases NSR. Both the distance cut-off and the L1 help the optimization by creating sparser matrices.

In fine, our second baseline reaches a mean NSR of 33.6%. We then used the median NSR as main metric and we have a median NSR of 34.9 %. This baseline is Effie V1.

C.2 Second Baseline: Distance Features

This section describes the adaptation made upon changing from quaternion to distance features. The resulting version, Effie V2.1, was then optimized in memory so that it could process proteins up to $10,000$ residues. The optimized version is Effie V2.2.

New Features

We replace the input features with distance features, and we kept the architecture and the optimization parameters unchanged. As shown in Table C.4, performances clearly improved.

Features	Accuracy	NSR
Relative position	40.4% %	34.9%
Distances	**43.9%**	**38.2%**

Table C.4: Comparison of input features.

Further investigation, displayed in table C.5, shown that adding information about dihedral angles was not useful. It must be implied by the set of distances given as input. We also tried to set a distance cut-off to define which neighbours are within the environment (parameter `thresh_neigh`), which enhanced accuracy. In our final architecture, this threshold is replaced by a sequence length of 128, which is more then the contact number at 10Å, ensuring that all the neighbours within 10Å are considered.

Dihedral	Thresh_neigh	Accuracy	NSR
No	None	43.9%	38.2%
Yes	None	41.8%	-
Yes	10	43.6%	37.9%

Table C.5: Influence of dihedral angles and distance cut-off on the neighbourhood of a target residue.

Adding the E-NPLL

Event though the NPLL was developed to tackle logical information, it proved to be beneficial in the case of proteins as well. As displayed in Table C.6, masking 10% of amino acids throughout training lead to a gain of 1.2% NSR.

E-NPLL	NSR
$p = 0$	38.2%
$p = 0.1$	**39.4%**

Table C.6: Using the E-NPLL instead of the regular NPLL improves NSR.

Alternative Training Schemes

The training schemes are detailed in Section 4.3. Here we report only results.

"Soft" PLL

Using a soft PLL to better learn degeneracy (the fact that several sequences fit the same backbone) resulted in deteriorated performances.

NPLL	Accuracy	NSR
Regular	**43.9%**	**38.2%**
Soft	41.9%	35.5%

Table C.7: Comparison of the soft NPLL with the regular.

Denoising Model

Multi-task training on denoising the input coordinate (measured by MSE loss) along the NPLL training resulted in deteriorated performances.

Loss	Accuracy	NSR
NPLL	**43.9%**	**38.2%**
NPLL+MSE	41.2%	34.1%

Table C.8: Comparison of the NPLL alone and multi-task training with the NPLL and a denoiser.

Adaptative Learning Rate

On the single-chain dataset, using an adaptive learning rate (LR) is detrimental. In Table C.9, 0 means there is no adaptive LR, 1 means the LR is proportional to our batch size (*i.e.*, the number of variables) and $\frac{1}{2}$ means the LR is proportional to the square root of the batch size. Interestingly, an adaptive LR is beneficial on the multi-chain dataset, that contains much larger proteins (see Annex C.3).

Reducing the Number of Parameters

Once parameters had been tuned, we realized our architecture had much more parameters than needed, which may lead to overfitting. Thus, we explored how to reduce some parameters. In these experiments, p is fixed to 0.1.

Adapt_LR	Acc	NSR
1/2	44.9%	39.7%
0	44.9%	**40.3%**
1	41.6 %	39.5%

Table C.9: Effect of using an adaptive learning rate on the single-chain dataset.

Detailed results are in Table C.10. We reduced both the depth and width of the resMLP (3 blocks ans 128 hidden neurons). For the gMLP, only the width is reduced. Using fewer kernels to encode the distance features degrades performances.

Description	# param	Acc	NSR	Conclusion
Baseline	5.2M	46.0	39.4	Baseline
nb_blocks = 3	1.5M	46.1	39.6	3 blocks are enough
depth_gMLP=10	4.5M	45.9	38.9	keep depth_gMLP=12
nb_kernel=8	5.1M	44.9	38.4	keep 16 kernels
ff_mult=4	4.5M	46.0	39.2	keep ff_mult=4
hidden=128, nb_blocs = 5	2.6M	46.3	39.5	Reduce hidden to 128
hidden=128, nb_blocs = 3	2.5M	46.2	**40.0**	new baseline
hidden=128, nb_blocs = 1	2.5M	45.4	39.4	Keep 3 blocks

Table C.10: Models with reduced number of parameters.

Effect of Symmetrization

We symmetrized the output of the neural net to interpret it as a graphical model. Yet, we found this is actually a beneficial inductive bias, that led to enhanced performances (Table C.11.)

Symmetrize	Accuracy	NSR
No	47.1%	36.4%
Yes	46.0%	39.4%

Table C.11: Effect of symmetrization.

Choice of the Value of p

We trained the same architecture with different value of the Emmental-PLL's parameter p. Results are in Table C.12. As we favour NSR over accuracy, we set $p = 0.3$.

Description	Acc	NSR median
p=0.1	46.0	39.4
p=0.2	45.1	39.7
p=0.3	44.8	40.5
p=0.4	44.7	40.3

Table C.12: Performances of the same architecture trained with different value of k.

C.3 Extension to Multi-chain Proteins

On the multi-chain dataset, we hypothesized that a higher capacity was required to capture both inter and intra-chain interactions. In Table C.13, models are assessed on the multi-chain validation set, therefore results cannot be directly compared with those obtained previously on the single-chain. Moreover, for a fair comparison between models, we randomly selected 100 protein structures in the validation set together with one sequence, and we computed metrics on this subset.

Increasing the Capacity of the Neural Nets

We decided to increase both the width and the depth of the resMLP (to 10 blocks and 258 hidden neurons) and the depth of the gMLP (to 15). The output dimension of the gMLP was increased to 128, but its width was kept the same, which was convenient as it limits the memory usage. In all these models, the initial learning rate was dropped to 5.10^{-5} as it leads to better validation loss.

Description	k	Acc	NSR	Conclusion
Baseline	False	51.9%	-	-
hidden = 256, n_block=5, gMLP_outputdim=128	10	52.7%	-	Keep all modifs
same + ff_mult=6	10	52.5%	-	Keep ff_mult=4
same + ff_mult=6 + depth_gMLP=15	10	53.4%	-	Keep depth gMLP 15
same + gMLPout 128	10	53.9%	46.1%	Keep gMLPout=128
same + n_blocks=10	10	54.6%	46.9%	Keep 10 blocks
same	30	54.0%	47.6%	k=30

Table C.13: Increasing the capacity for multi-chain proteins.

Adaptive Learning Rate

Multi-chain proteins being much larger than single-chain proteins, training with an adaptive learning rate becomes beneficial, as shown in Table C.9.

Adapt LR	Acc	NSR
1/2	54.6%	**46.9%**
0	51.0%	45.0%

Table C.14: Comparison of model with an adaptive learning rate proportional to the square root of the batch size (1/2) or with no adaptive LR (0).

Noisy Inference and Forward Folding

Noise (train)	Noise (inf)	NSR (median)	plddt (mean)	pTM (mean)
0.2	0	39.0	60.0	0.41
0.2	0.2	35.0	56.7	0.37
0.02	0	44.5	52.6	0.35
0.02	0.02	44.0	53.2	0.36

Table C.15: NSR and forward folding metric on models noised during training. Adding noise during inference (inf) as well deteriorates metrics.

Appendix D

Effie Relaxation

In practical design, the optimization of a sequence is always followed by a minimization or relaxation procedure, usually using RosettaDesign's tools, to allow the structure to make small conformationnal adjustments. One design protocol of Rosetta, RosettaFastDesign, consists in successive rounds of design plus relaxation, with the relaxed structure being the input for the successive round. Inspired by this methodology, we tried to obtain an end-to-end relaxation of Effie through the neural architecture.

The forward pass of our neural model map coordinates to cost functions, which, once optimized, produce a sequence. The idea is to freeze the neural network and to backpropagate to update the coordinates crd such that they are more suited to the sequence y. Both the native or a designed sequence can be backpropagated.

$$crd = crd - \eta \frac{\partial PLL(\mathcal{N}(crd), y)}{\partial crd}$$

With η a parameter to control the range of the coordinate change (similarly to the learning rate).

We directly applied an adversarial method, the Fast Gradient Sign Attack [Goodfellow et al., 2014].

Constraining the Coordinate Updates

The issue with this approach is that coordinates are updated without any knowledge of what the input represent. However, coordinates represent atoms that are linked together by covalent bonds, and therefore they cannot move independently. The expected consequence is that the point cloud resulting from a coordinate update does not resemble a protein anymore. Indeed, we report the span on the value of

bond length and angles before and after relaxation in Table D.1, and we notice the span is much wider after relaxation.

Bond/angle	crd (PDB)	crd (relaxed)
N-C_α (Å)	1.43-1.49	0.89-3.14
C_α-C (Å)	1.48-1.57	0.66-3.85
ω (°)	162-201	98-278

Table D.1: Span on two bond length and the angle ω observed before and after relaxation ($\eta = 0.01$) on the protein 1A2F, chain A.

We listed 5 constraints on each residue the resulting structure should satisfy. Four are on bond angles between each atom, that is known to be approximately constant. The last one is on the dihedral angle ω that is planar: most of the time $\omega = 180°$ (trans-residue), and some times $\omega = 0°$ (cis-residue). More precisely, we set (length are in Å):

$$\begin{aligned} bond\_NC_\alpha &= (71 + 75)/100 \\ bond\_CC &= (76 * 2)/100 \\ bond\_CO &= (67 + 57)/100 \\ bond\_CN &= 1.32 \end{aligned}$$

We did not enforce those constraint directly, but we rather added a penalization term to the loss evaluating how violated a constraint is. This term is an harmonic potential on each bond and ω angle, *i.e.,* the square difference between expected and observed value of the bond/angle. We set the weight of this penalization to $\frac{0.1}{\eta}$. As displayed in Figure D.1, the penalization term is enough to maintain plausible structures throughout several rounds of relaxation. Moreover, even though we did not constraint ϕ and ψ dihedral angles, they keep plausible values through relaxation, as displayed is the Ramachandran plot in Figure D.2.

Testing the Relaxation

Our first test was on a xylanase that was previously redesigned by the team (PDB id 2C1F). We first perturbed its coordinates with a Gaussian noise, then we applied several iterations of relaxation (by backpropagating the native sequence) and we found that the relaxed structure was closer to the initial structure than the perturbed structure (in terms of RMSD, see Figure D.3). We then tried to mutate some residues in loops, either a small one into a large one (*e.g.,* G129Y or G129F) or vice versa (*e.g.,* W188A or W188G), to see if the structure was relaxed to accommodate the

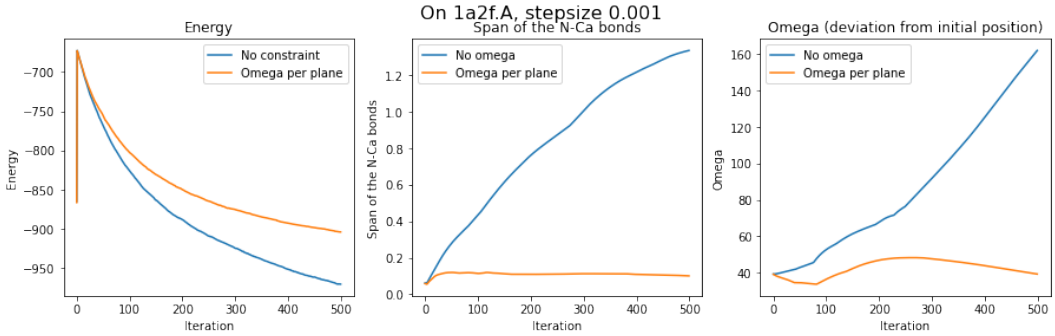


Figure D.1: Variation of the predicted energy, the span of the $N - C_\alpha$ bond and of the deviation of ω angle form its initial position through several steps of relaxation, with or without penalization term to encore constraints.

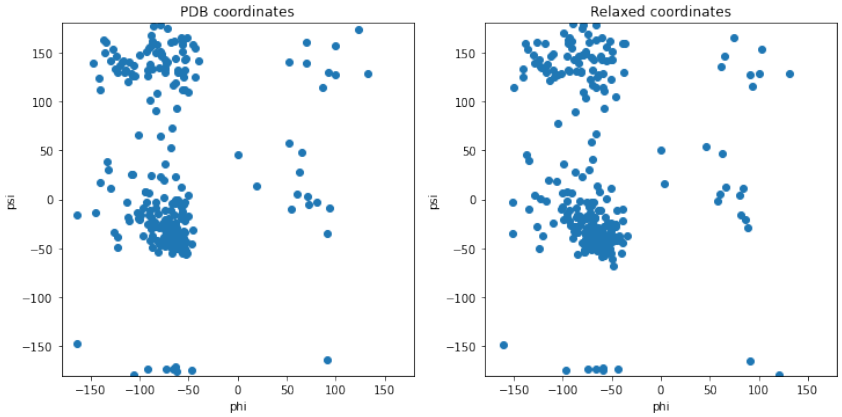


Figure D.2: Ramachandran plots of dihedral angles in the protein 1A2F, before and after relaxation.

change in residue size. If Effie always disfavoured the mutation with respect to the native residue, inspection of the resulting structure with PyMol was not conclusive.

Finally, we used the relaxation to carry out our version of Rosetta FastDesign protocol. Using Effie (V2.1-single), we iteratively designed the proteins of the single-chain validation set: on sequence was predicted, then it was backpropagated to relax the structure, which was design again, n times in total. For the n designed sequences, the one with the lowest predicted score was chosen. Then we compared the resulting NSR to the one obtained by our standard design procedure (*i.e,* $n = 1$). We varied the parameters n and η, but each time we obtained a median gain in NSR very close to 0. Thus, the iterative relaxation+design does not seem to improve the quality of designed proteins. It might be interesting to complement this observation with forward-folding metrics.

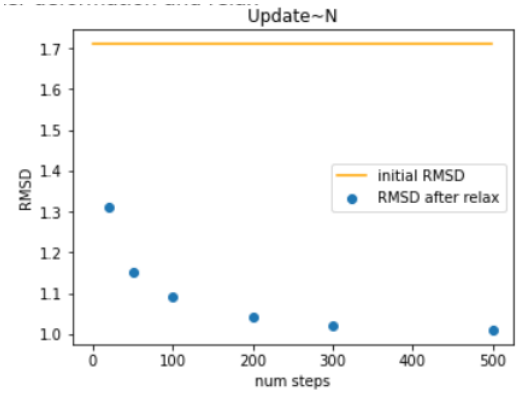


Figure D.3: Relaxation of a perturbed backbone, and its alignment to the original structure through the relaxation (measured in RMSD).

Appendix E

Résumé en français

Contexte

Les protéines sont des molécules complexes essentielles à la vie. En effet, elles sont en charge de nombreux processus au sein de cellules: elles fournissent une structure, catalysent des réactions, transmettent des signaux et bien plus encore. Cette grande variété de fonctions peut être réutilisée dans de nombreuses applications en biotechnologie, médecine, chimie verte, etc. Si des millions d'années d'évolution ont adapté les protéines pour améliorer leur fonction ou en créer de nouvelles pour les besoins biologiques, les applications industrielles présentent des conditions spécifiques pour lesquelles les protéines naturelles peuvent ne pas convenir. Le design computationnel de protéines (CPD) vise à trouver de nouvelles protéines ayant les propriétés ou les fonctions souhaitées [Huang et al., 2016].

Les protéines sont composées d'une succession de petites molécules appelées acides aminés. La plupart des protéines se replient selon une forme 3D spécifique et déterminée par les propriétés physico-chimiques de leurs acides aminés. La fonction d'une protéine étant reliée à sa structure 3D, le CPD peut consister à trouver une séquence se repliant sur une structure cible choisie pour porter les caractéritiques souhaitées. La recherche de séquences peut être reformulée comme un problème de raisonnement discret dont le but est de minimiser une fonction de score capturant les interactions au sein de la protéine. Les fonctions de score existantes sont basées sur des approximations physiques et/ou des statistiques [Park et al., 2016] et leur qualité peut être limitante en pratique.

Problématique

Dans cette thèse, nous cherchons à capturer plus finement la relation séquence-structure des protéines naturelles en apprenant une nouvelle fonction de score à l'aide du Deep Learning ("apprentissage profond"). Cette fonction de score est

optimisée avec les outils de raisonnement discret existants [Traoré et al., 2013] pour designer de nouvelles protéines. Elle peut optionellement être combinée avec des contraintes ou des connaissances supplémentaires pour mieux guider le design vers une séquence de protéine présentant toutes les caractéristiques voulues.

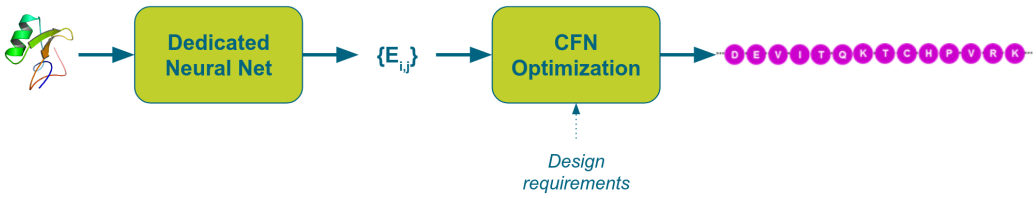


Figure E.1: Vue schématique de la pipeline que nous proposons pour le design computationnel de protéines. $E_{i,j}$ est la fonction de score entre deux amino acides i et j de la protéine.

Notre objectif se heurte à deux difficultés principales. Premièrement, les structures de protéines sont des données non-Euclidiennes et donc difficiles à représenter en vue de leur traitement par un réseau de neurones. Deuxièmement, nous voulons développer un pipeline hybride capable à la fois de raisonner et d'apprendre. La construction d'un tel composé hybride est l'un des défis ouverts de l'intelligence artificielle [Hochreiter, 2022]. Dans cette thèse, nous abordons les deux difficultés séparément.

Contributions

Ces travaux sont organisés en 5 chapitres et autour de 3 contributions principales: une classification des approches de CPD basées Deep Learning; une fonction de perte ("loss") pour combiner apprentissage et raisonnement; une fonction de score apprise pour le design de protéines.

Nous avons commencé par introduire toutes les notions utilisées dans ces travaux trans-disciplinaires dans le chapitre 1: les protéines, le Deep Learning et les modèles graphiques, qui sont le modèle de raisonnement que nous utilisons. Nous avons ensuite répertorié les approches existantes pour le CPD, avec une attention particulière sur les méthodes les plus récentes basées sur le Deep Learning [Ingraham et al., 2019; Dauparas et al., 2022]. Dans une revue publiée en 2021 [Defresne et al., 2021] et complétée dans le chapitre 2, nous avons classé ces méthodes selon la représentation des protéines utilisée, et nous avons montré qu'il n'y a pour l'instant pas de consensus sur la manière la plus adaptée de représenter une structure de protéine pour la tache de design. Nous avons également comparé ces méthodes utilisant du Deep Learning avec les méthodes plus traditionnelles optimisant une fonction de score. Les premières offrent de meilleures performances, à la fois computationnellement et expérimentalement, mais les dernières présentent des avantages

pratiques puisqu'elles permettent d'ajouter des objectifs de design et/ou des contraines supplémentaires. Nos travaux cherchent à réunir le meilleur des deux mondes dans une méthode hybride consistant à apprendre une fonction de score, puis à l'optimiser pour designer de nouvelles protéines.

Le chapitre 3 a été l'occasion de s'éloigner momentanément des protéines pour se concentrer sur l'intelligence artificielle hybride. Une brève description des méthodes actuelles nous a permis de placer nos travaux dans le cadre du Decision-Focused Learning ("apprentissage axé sur la prise de décision"). Notre formulation du problème de CPD nécessite une approche hybride qui s'adapte à de grandes instances, car les protéines peuvent contenir des milliers d'acides aminés. Nous voulons aussi bénéficier d'une résolution exacte à l'inférence. Puisque qu'aucune approche existante n'offre les deux avantages, nous avons développé une methode basée sur une nouvelle fonction de perte, la pseudo log-vraissemblance emmental [Defresne et al., 2023]. Ces travaux ont été publiés et présentés à IJCAI 2023. Pour développer cette approche, nous avons travaillé sur l'apprentissage des règles du Sudoku, un problème-jouet avec d'intéressants parallèles avec le CPD. Notre fonction de perte nous a permis de dépasser l'état de l'art sur ce problème et quelques variants, qui sont un benchmark usuel des methodes pour apprendre à raisonner. Enfin, apprendre à jouer au Sudoku a été l'occasion d'illustrer les avantages de combiner raisonnement et apprentissage. En effet, cela est plus interprétable, nécessite moins de données et offre la possibilité d'ajouter des connaissances ou contraintes *a posteriori*. Toutes ces propriétés sont également souhaitables pour le CPD.

Nous sommes revenus à notre problème de design de protéines au chapitre 4, où nous avons présenté la représentation de protéines que nous avons choisie, basée sur des paires d'acides aminés. Nous avons également détaillé toute l'architecture utilisée pour apprendre Effie, notre fonction de score sous la forme d'un modèle graphique pour le design de protéines. Nous l'avons optimisée pour designer des protéines puis nous l'avons evaluée de manière approfondie *in silico* (*i.e.*, computationellement) dans le chapitre 5. L'optimisation d'Effie permet d'obtenir des séquences de meilleure qualité qu'avec les fonctions de score traditionnelles, tout en étant compétitive avec les méthodes utilisant seulement du Deep Learning. De plus, Effie peut également être utilisée sur des taches pour lesquelles elle n'a pas été entraînée, suggérant que des concepts physico-chimiques ont été appris. Enfin, nous avons montré les avantages de notre méthode hybride par rapport aux méthodes purement Deep Learning sur trois projets appliqués nécessitant l'ajout de contraintes ou de connaissance et l'optimisation des séquences designées. L'un de ces projets a été validé expérimentalement.

Bibliography

Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Courbet, A., de Haas, R. J., Bethel, N., Leung, P. J. Y.,

Huddy, T. F., Pellock, S., Tischer, D., Chan, F., Koepnick, B., Nguyen, H., Kang, A., Sankaran, B., Bera, A. K., King, N. P., and Baker, D. (2022). Robust deep learning-based protein sequence design using proteinMPNN. *Science*, 378(6615):49–56.

Defresne, M., Barbe, S., and Schiex, T. (2021). Protein design with deep learning. *International Journal of Molecular Sciences*, 22(21):11741.

Defresne, M., Barbe, S., and Schiex, T. (2023). Scalable coupling of deep learning with logical reasoning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 3615–3623.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Hochreiter, S. (2022). Toward a broad ai. *Communications of the ACM*, 65(4):56–57.

Huang, P., Boyken, S., and Baker, D. (2016). The coming of age of de novo protein design. *Nature*, 537(7620):320–327.

Ingraham, J., Garg, V. K., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

Park, H., Bradley, P., Greisen, P., Liu, Y., Mulligan, V. K., Kim, D. E., Baker, D., and DiMaio, F. (2016). Simultaneous optimization of biomolecular energy functions on features from small molecules and macromolecules. *Journal of Chemical Theory and Computation*, 12(12):6201–6212.

Traoré, S., Allouche, D., André, I., de Givry, S., Katsirelos, G., Schiex, T., and Barbe, S. (2013). A new framework for computational protein design through cost function network optimization. *Bioinformatics*, 29(17):2129–2136.

Abstract. Proteins are complex molecules that fulfill many functions in living organisms. Some of these functions can be repurposed for applications in biotechnology, medicine, green chemistry .... The goal of Computational Protein Design (CPD) is to predict a protein sequence fit for an application. Since the function of a protein is tightly linked to its 3D structure, CPD can be formulated as predicting a sequence folding onto a target structure and therefore fulfilling a function of interest. Existing approaches are based on the optimization of an energy function scoring interactions within the proteins or they are purely based on Deep Learning. In this thesis, we present a new hybrid approach for CPD, combining Deep Learning (DL) and Automated Reasoning.

Our first contribution is to categorized existing DL approaches based on the protein representation. Discussing their advantages and drawbacks with respect to traditional energy-based methods lead us to try and take the best of both worlds by learning a new scoring function that is optimized to design proteins. This score function is a Graphical Model, a reasoning compound already successfully used to optimize proteins. This objective requires a hybrid pipeline combining Deep Learning and discrete optimization. Such hybridization being an open challenge in Artificial Intelligence, we first developed a method to learn Graphical Models from data that allows exact inference while being scalable. It was developed on the standard benchmark of learning how to play Sudoku, on which it reaches state-of-the-art results.

We then applied this hybrid pipeline to protein design. A protein structure being non-Euclidean data, it requires a suited representation and a fitting neural architecture to be processed. We learned a new scoring function for design that we named Effie. We extensively validated it *in silico*. On design tasks, it outperformed traditional energy-based methods while being competitive with DL-based approaches. Moreover, it can tackle tasks for which it has not been explicitly trained, suggesting some physical-chemical concepts have been learned. Finally, we applied it on 3 projects where the design objectives required to bias or conditioned Effie *a posteriori* via the addition of knowledge or constraints. In this context, we showed the interest of our hybrid approach as Effie + discrete optimization outperformed pure Deep Learning methods.

Keywords. Computational Protein Design, Deep Learning, Automated Reasoning, Hybrid AI, Graphical Model.

Résumé. Les protéines sont des molécules complexes qui remplissent de nombreuses fonctions dans les organismes vivants. Certaines de ces fonctions peuvent être reprises pour des applications en biotechnologie, médecine, chimie verte, etc. L'objectif du design computationnel de protéines (CPD) est de prédire une séquence de protéine adaptée à une application. La fonction d'une protéine étant étroitement liée à sa structure 3D, le CPD peut être formulé comme la prédiction d'une séquence se repliant sur une structure cible et remplissant ainsi la fonction d'intérêt. Les approches existantes sont basées soit sur l'optimisation d'une fonction d'énergie évaluant les interactions au sein d'une protéine, ou sont soit purement basées sur l'apprentissage profond. Dans cette thèse, nous présentons une nouvelle approche hybride pour le CPD, combinant Deep Learning (DL) et raisonnement automatique.

Notre première contribution consiste à catégoriser les approches DL existantes selon la représentation des protéines utilisée. Discuter de leurs avantages et inconvénients par rapport aux méthodes traditionnelles basées sur l'énergie nous a conduits à vouloir essayer de prendre le meilleur des deux mondes en apprenant une nouvelle fonction de score optimisée pour la conception de protéines. Cette fonction de score est un modèle graphique, un composant de raisonnement déjà utilisé avec succès pour optimiser des protéines. Notre objectif nécessite une pipeline hybride combinant Deep Learning et optimisation discrète. Une telle hybridation étant un défi ouvert en Intelligence Artificielle, nous avons d'abord développé une méthode pour apprendre un modèle graphique à partir de données et qui permet une inférence exacte tout en passant à l'échelle sur de grandes instances. Cette méthode a été développée sur le benchmark standard de l'apprentissage des règles du Sudoku, sur lequel elle dépasse l'état de l'art.

Nous avons ensuite appliqué cette architecture hybride à la conception de protéines. La structure d'une protéine étant une donnée non euclidienne, elle nécessite une représentation adaptée et une architecture neuronale adéquate pour être traitée. Nous avons appris une nouvelle fonction de score pour la conception que nous avons appelée Effie. Nous l'avons d'abord validée *in silico*. Pour les tâches de design, elle surpasse les méthodes traditionnelles basées sur l'énergie tout en étant compétitive par rapport aux approches basées DL. De plus, elle peut s'attaquer à des tâches pour lesquelles elle n'a pas été explicitement entraînée, ce qui suggère qu'elle a appris certains concepts physico-chimiques. Enfin, nous l'avons appliquée sur 3 projets concrets dont les objectifs de design nécessitaient de biaiser ou de conditionner Effie *a posteriori* via l'ajout de connaissances ou de contraintes. Dans ce contexte, nous avons montré l'intérêt de notre approche hybride puisque Effie + optimisation discrète a surpassé les méthodes de Deep Learning pures.

Mot-clés. Design Computationnel de Protéines, Deep Learning, Raisonnement Automatique, IA hybride, Modèles Graphiques.