Contents

## 1. Introduction

Proteins are biomolecules that are fundamental to nearly all biological processes. Their diverse roles include transporting nutrients, catalyzing chemical reactions, providing structural support, and more. The function of a protein is determined by the composition of its primary sequence created from amino acids. A single protein sequence can vary greatly in length and order of its amino acids, leading to a very large number of possible configurations.

The size of the protein sequence space makes exhaustive experimental exploration infeasible. However, the rapid growth of biological databases, driven by advances in sequencing technologies, has transformed biology into a data-rich discipline. Large repositories such as UniProt, Pfam, and the Protein Data Bank store millions of sequences and structures, providing crucial resources for computational approaches[1].

This wealth of data has enabled the development of advanced statistical and machine learning models capable of simulating protein sequence evolution, prediciting structural conformations, and generating novel sequences with desired properties. In addition, breakthroughs in protein structure prediction–most notably the Nobel Prize winning AlphaFold[2]– show that computational methods can rival experimental accuracy, in a much cheaper manner.

In this work, we will implement an efficient autoregressive network for protein sequence generation leveraging the Direct Coupling Analysis (DCA) in Python. In Section 2, the biological background will be laid out. Section 3 will introduce the mathematical foundations. Section 4 will detail previous iterations of similar models, while Section 5 will explore our implementation.

## 2. Biological Background

### 2.1. Proteins and their structure

Proteins are essential biological molecules responsible for a wide range of functions in living organisms. Despite their functional diversity, all proteins are polymers of the same set of standard building blocks: the 20 canonical amino acids arranged in different assortments. Amino acids share a common core structure consisting of a central carbon atom bonded to a hydrogen atom, an amino group, a carboxyl group, and a variable side chain. This side chain is the defining feature of each amino acid, giving rise to differences in size, shape, chemical reactivity, and polarity. Proteins are formed by peptide bonds of distinct amino acids, where the term polypeptides comes from. In this process, certain atoms are lost, thus, within a protein sequence, individual amino acids are typically referred to as residues. Generally, protein sequences are made up of between 50 and 2000 amino acids. The ordering of the amino acids dictates how a protein folds into its three-dimensional structure, known as its conformation. Although each conformation is unique, two common folding patterns occur in many proteins: the $\alpha$ helix and the $\beta$ sheet [3].

Each amino acid is chemically distinct and can occur at any position in a protein chain, giving rise to $20^n$ possible polypeptide sequences of length $n$. For a typical protein of 300 amino acids, the number of possible sequences is astronomically large ($\approx 2 \cdot 10^{390}$). However, only a small fraction of these sequences are capable of folding into a stable three-dimensional conformation.

### 2.2. Protein families and evolutionary information

Proteins do not evolve in isolation; they often belong to protein families, groups of proteins that share a common evolutionary origin, therefore exhibit similar sequence features and functional properties [4]. Over evolutionary timescales, mutations accumulate in these families: some are beneficial, altering the protein activity in ways that give rise to new functions, while many others are neutral and have no effect on stability

or activity. Harmful mutations, by contrast, disrupt protein folding or its function. These changes are eliminated through natural selection. The result is a collection of homologous proteins that retain overall structural and functional characteristics, but also display sequence variability that encodes the evolutionary history of the family [3].

The study of evolutionary history and relationships among biological entities is referred to as phylogenetics. Protein families provide the domain for phylogenetic analysis, as examining families provides insight that cannot be obtained from a single sequence[5]. From homologous proteins, we can detect important correlated mutations between amino acid positions which represents constraints enforced to maintain protein integrity by evolution. These statistical patterns are crucial for computational approaches as they attempt to learn them to predict three-dimensional structure and understand protein function.

### 2.3. Multiple sequence alignments

To extract important statistical information from protein families, the homologous sequences need to be organized in a systematic way. This is done through a multiple sequence alignment (MSA), in which three or more sequences are arranged so that homologous sites are placed in the same column [6]. Alignment gaps are introduced throughout the MSA to maximise the positional correspondence of sequences and enable them to have the same fixed length.

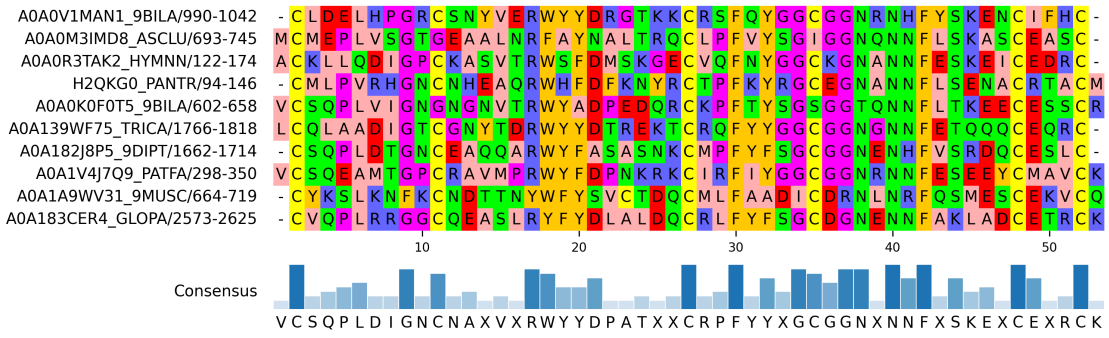


Figure 1: MSA of first 10 proteins in PF00014, created using `pyMSAviz`[7]

Aligning sequences in this manner, patterns of conservation and variation are revealed across the family. Conserved positions, those which are unchanged in multiple sequences, represent sites that are critical for maintaining structure or function, while variable positions indicate sites that can tolerate mutations without disruption to the protein. Beyond conservation, MSAs also capture covariation: pairs of positions that mutate in a correlated way across sequences. These covariation signals reflect couplings, where a mutation at one site requires compensation at another to maintain protein integrity [8].

### 2.4. Protein contacts

One of the earliest challenges for researchers in computational biology–historically encouraged by the CASP (Critical Assessment of Structure Prediction) competition–has been to understand how the linear sequence of amino acids folds into its conformation. Researchers explored spatially close pairs of residues in a protein's folded three-dimensional structure, called contacts. As the structures can be represented in Cartesian coordinates $(x, y, z)$, contacts are defined using distance thresholds. Two residues are generally considered to be in contact if the distance between the selected atoms is below a set threshold, usually 8 Ångströms (Å).

Residues that are far apart in the linear sequence, may be close together in the 3D structure. The contact distances are further categorized into short, medium, and long range predictions. Most computational approaches evaluate the long range contacts separately, as they are the most important for accurate predictions and unsurprisingly, the hardest to predict [9].

### 2.5. The problem of inference

A central challenge in protein sequence analysis is disentangling the true structural and functional constraints of protein families from spurious correlations introduced by other factors. Correlations can arise indirectly, for example, if residue A correlates with B, and B with C, then A and C may appear correlated without a direct interaction.

Moreover, shared evolutionary history (phylogeny) and biases in sequence databases can further obscure the true couplings that underlie protein folding and function [10].

## 3. Mathematical Foundations

### 3.1. Proteins as statistical systems

Protein sequences can be thought of as random variables produced by a certain distribution. Each sequence of length $L$ can be written as:

$$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, ..., \sigma_L), \quad \sigma_i \in \mathcal{A}, \tag{1}$$

where $\mathcal{A}$ is the alphabet of size $q = 21$ (20 amino acids and the alignment gap) and $\sigma_i$ are the residue sites. We can organize these sequences into multiple sequence alignments, a table $\left\{\boldsymbol{\sigma}^{(m)}\right\}_{m=1}^{M}$ of $M$ empirical samples. These samples are aligned to have a common length $L$. Each row in the MSA represents a protein, and each column a position, or site, in the sequence. From the alignments, we can define single and pairwise frequency counts for the columns. The single-site frequency for MSA column $i$ can be computed as:

$$f_i(A) = \frac{1}{M}\sum_{m=1}^{M}\delta\left(\sigma_i^{(m)}, A\right), \quad A \in \mathcal{A}, \quad \text{where } \delta \text{ is the Kronecker delta}^1 \tag{2}$$

while the pairwise frequency of MSA columns $i, j$ is computed as:

$$f_{ij}(A, B) = \frac{1}{M}\sum_{m=1}^{M}\delta\left(\sigma_i^{(m)}, A\right)\delta\left(\sigma_j^{(m)}, B\right) \tag{3}$$

The empirical frequencies will serve as constraints for the distribution we want to infer.

### 3.2. Maximum Entropy Principle

To find the probability distribution $P(\boldsymbol{\sigma})$ that will satisfy constraints, in other words reproduce the empirical marginals $f_i(A)$ and $f_{ij}(A, B)$, we can use the maximum

[1] $\delta(x, y) := \begin{cases} 0 \text{ if } x \neq y \\ 1 \text{ if } x = y \end{cases}$

entropy principle[11], [12]. The first step to set up the MEP is extracting information from the system. Usually, this information is given in the form of averages of functions $\langle f_k(x) \rangle$. For example, in a physical system, one could compute average magnetization or energy of the observed system. We also need to define a probability of occupancy of states, $p(x)$, which runs over all the possible states. This distribution has the usual property of mapping each state to a value within 0 and 1 and adding up to 1 when considering all states [13]. Our uncertainty on the system is expressed quantitatively through Shannon entropy $S$[14]:

$$S = -\sum_x p(x) \ln p(x) \tag{4}$$

Generally, the distribution which maximizes entropy is the uniform distribution. However, since constraints affect the probabilities of states, the uniform is not suitable. The constraints on our system are:

$$\sum_x p(x) = 1, \quad \text{and} \quad \sum_x p(x) f_k(x) = \langle f_k \rangle \quad (k = 1, ..., m). \tag{5}$$

Selecting Shannon entropy as our measure of information, we can introduce Lagrange Multipliers $\lambda_0, \lambda_1, ...\lambda_m$ to maximize Eq. 4 subject to the constraints Eq. 5, yielding:

$$p(x) = \exp\left( -\lambda_0 - \sum_{k=1}^{m} \lambda_k f_k(x) \right). \tag{6}$$

Define the partition function $Z$ as:

$$Z(\lambda) = \sum_x \exp\left( -\sum_{k=1}^{m} \lambda_k f_k(x) \right). \tag{7}$$

With normalization $\lambda_0 = \ln Z$, we can write the moments as:

$$\langle f_k \rangle = -\frac{\partial}{\partial \lambda_k} \ln Z(\lambda) \tag{8}$$

The entropy of the distribution then reduces to[2]:

$$S_{\max} = \lambda_0 + \sum_{k=1}^{m} \lambda_k \langle f_k \rangle \tag{9}$$

The maximum-entropy (MaxEnt) distribution for this system is exactly the Potts model.

### 3.3. Connection to statistical mechanics

The Potts model is rooted in statistical mechanics, where it was introduced as a generalization of the Ising spin model. Both models were developed to study ferromagnetism and phase transitions.

#### 3.3.1. Ising Model

The Ising model [15] describes a system of spins $\sigma_i$ arranged on a lattice. Each spin can take one of two possible values:

$$\sigma_i \in \{+1, -1\} \tag{10}$$

The Hamiltonian function, which represents the energy, of a spin configuration $\{\sigma_i\}$ is

$$H_{\text{Ising}}(\{\sigma\}) = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - h \sum_i \sigma_i, \tag{11}$$

where $J$ is the coupling constant calculated over nearest-neighbor pairs (represented by $\langle i, j \rangle$) that defines the strength between paired interactions. The $h$ term is the external magnetic field acting on the spins.

At thermal equilibrium, the probability of a configuration is given by the Boltzmann distribution:

[2]The full derivation can be found in Appendix A.

$$P(\{\sigma\}) = \frac{1}{Z}\exp\big(-\beta H_{\text{Ising}}(\{\sigma\})\big), \tag{12}$$

with $\beta = 1/(k_b T)$ and partition function

$$Z = \sum_{\{\sigma\}} \exp\big(-\beta H_{\text{Ising}}(\{\sigma\})\big). \tag{13}$$

### 3.3.2. Potts Model

The Potts model [16] generalizes the Ising model by allowing each spin to take on $q$ possible states. The $q$-state Potts model states take values in $\{1, 2, ...q\}$. The Hamiltonian is written as

$$H_{\text{Potts}}(\{\sigma\}) = -J \sum_{\langle i,j \rangle} \delta\big(\sigma_i, \sigma_j\big). \tag{14}$$

$J$ again represents the ferromagnetic coupling that encourages neighboring spins to align in the same state. The original Potts was defined without the local fields, but they can be added if needed. The partition function becomes

$$Z = \sum_{\{\sigma\}} \exp(-\beta H_{\text{Potts}}(\{\sigma\})). \tag{15}$$

The Potts model simplifies to an Ising model when $q = 2$. Putting this together, the probability distribution from the Potts model is written as:

$$P(\boldsymbol{\sigma}) = \frac{1}{Z}\exp(-\beta H_{\text{Potts}}(\{\sigma\})). \tag{16}$$

It is important to note–anticipating further exploration–that the Potts model is over-parametrized: many different $(h, J)$ sets define exactly the same distribution. Without a gauge choice, the parameters will not be uniquely identifiable and the norms of $J$ can be misleading, hindering the process of optimization. Some common gauges, which will

be explored in detail further, are reference states, zero-sum, and defining it implicitly via regularization.

### 3.4. Direct Coupling Analysis

The statistical dependencies observed in MSAs encode important information about protein structure and function. However, naive measures such as covariance cannot distinguish between direct correlations, which arise from true physical or functional contacts, and indirect correlations, propagated via other residues. Direct Coupling Analysis (DCA) addresses this problem by introducing a method to separate direct correlations.

DCA assumes that sequences in the MSA are independent realizations of a Potts model distribution:

$$P(\boldsymbol{\sigma}) = \frac{1}{Z} \exp\left( \sum_i h_i(\sigma_i) + \sum_{i<j} J_{ij}(\sigma_i, \sigma_j) \right) \tag{17}$$

where $h_i$ are site-specific fields and $J_{ij}$ are coupling parameters between residue pairs. The central goal of DCA is to infer the interaction parameters $J_{ij}$ that best explain the observed single- and pairwise frequencies. In practice, direct inference of these parameters is computationally challenging. Evaluating the partition function $Z$ is intractable for realistic proteins, as it requires summing over $q^L$ possible sequences. Thus, distinct implementations of DCA introduce approximations to bypass this computation. The original message passing implementation [17], directly attempted to solve for couplings through a slowly converging iterative scheme, making it inapplicable in practice. In this paper, we will explore some important algorithmic innovations in DCA following its conception.

## 4. Evolution of DCA Methods

### 4.1. Mean-field DCA (2011)

The mean-field Direct Coupling Analysis (mfDCA) algorithm, introduced by Morcos et al. [18], provided the first computationally feasible approximation of the Potts model. The idea is to approximate a small-coupling expansion, valid when correlations between sites are weak which reduces the inference problem to the inversion of a correlation matrix.

#### 4.1.1. Method

To begin, the raw frequency counts suffer from sampling bias, so the weight of highly similar sequences is reduced. Each sequence $\sigma^{(a)}$ is assigned a weight

$$w_m = \frac{1}{\left|\{b \in \{1, ..., M\} : \mathrm{sim}(\sigma^{(a)}, \sigma^{(b)}) \geq x\}\right|}, \tag{18}$$

where M is the number of sequences in the MSA and sim is their similarity[3]. The $x$ represents the sequence identity threshold, which was empirically found to be best at $x = 0.8$. The effective weight of a sequence $m$ is $w_m$, and the total effective number of sequences is

$$M_{\text{eff}} = \sum_{m=1}^{M} w_m. \tag{19}$$

From these weights, the new regularized empirical single and pairwise frequencies are computed as

[3]The original paper used "seqid" to represent percentage identity. We chose "sim" as the future methods adopted this notation.

$$
\begin{aligned}
f_i(A) &= \frac{1}{M_{\text{eff}} + \lambda} \left( \frac{\lambda}{q} + \sum_{m=1}^{M} w_m \delta\Big(A, \sigma_i^{(m)}\Big) \right) \\
f_{ij}(A,B) &= \frac{1}{M_{\text{eff}} + \lambda} \left( \frac{\lambda}{q^2} + \sum_{m=1}^{M} w_m \delta\Big(A, \sigma_i^{(m)}\Big) \delta\Big(B, \sigma_j^{(m)}\Big) \right)
\end{aligned}
\tag{20}
$$

where $q = 21$ is the amino acid alphabet with the gap symbol, and $\lambda$ is a pseudocount parameter used for regularization. The gauge choice employed in mfDCA is defining a reference state to remove redundancy among parameters and ensure identifiability, while preserving all physically relevant couplings. Typically, it is set as the last amino acid $A = q$. This gives us

$$
\forall i, j : J_{ij}(a, q) = J_{ij}(q, a) = h_i(q) = 0 \tag{21}
$$

**Finding the Couplings**

To circumvent the intractable $Z$ computation, mfDCA assumes weak correlations between sites, expanding the exponential in Eq. 17 by the Taylor series to first order[19], [20]. This results in the relation

$$
C_{ij}(A, B) = f_{ij}(A, B) - f_i(A) f_j(B). \tag{22}
$$

between the couplings and the connected correlation matrix. The couplings are then approximated as through naive mean-field inversion

$$
e_{ij}(A, B) = -(C^{-1})_{ij}(A, B), \tag{23}
$$

where $C$ is treated as a $((q-1)L) \times ((q-1)L)$ matrix, and the pair $(i, A)$ is regarded as a single index. The full derivation can be found in Appendix B. In practice, the correlation matrix is often singular without regularization. mfDCA opts to use a strong pseudocount ($\lambda \approx M_{\text{eff}}$) to stabilize the inversion and prevent spurious large couplings.

**Pair Scoring**

Once the estimate of the pair couplings $e_{ij}(A, B)$ is inferred, we need a way to rank residue pairs by their interaction strength. The $(q-1) \times (q-1)$-dimensional coupling matrices need to map to a single scalar parameter. We can do this through the direct information (DI) [17]. To begin, for each pair $(i, j)$ a two-site model is constructed:

$$P_{ij}^{(\mathrm{dir})}(A, B) = \frac{1}{Z_{ij}} \exp\big(e_{ij}(A, B) + \tilde{h}_i(A) + \tilde{h}_j(B)\big), \tag{24}$$

with auxiliary fields $\tilde{h}_i, \tilde{h}_j$ chosen such that

$$\sum_{B=1}^{q} P_{ij}^{(\mathrm{dir})}(A, B) = f_i(A), \quad \sum_{A=1}^{q} P_{ij}^{(\mathrm{dir})}(A, B) = f_j(B). \tag{25}$$

The direct information is the mutual information associated to this distribution:

$$\mathcal{S}_{ij}^{\mathrm{DI}} = \sum_{A,B=1}^{q} P_{ij}^{(\mathrm{dir})}(A, B) \ln \frac{P_{ij}^{(\mathrm{dir})}(A, B)}{f_i(A) f_j(B)} \tag{26}$$

The top-scoring pairs are predicted to be structural contacts.

#### 4.1.2. Limitations

While mfDCA represented a breakthrough in usability of DCA, the simplifying approximations impose limitations on the model:

- Weak coupling assumptions: the small-coupling expasion assumes nearly linear correlations, which can underestimate strong epistatic effects in proteins [21].
- Computational scaling: the inversion of the correlation matrix scales as $\mathcal{O}(L^3)$, which is costly for very large MSAs.
- Pseudocount dependence: due to the algorithm's vast parameter size (in the order of $400N^2$) strong regularization is required. This makes the choice of the pseudocount $\lambda$ significantly affect performance.

### 4.2. Pseudo-likelihood Maximization DCA (2013)

While mfDCA provided a fast approximation, it relied on strong assumptions. Pseudolikelihood maximization (plmDCA) relaxed these by fitting sitewise conditional probabilities directly. Concretely, the inverse Potts problem on an alignment of length $L$ becomes $L$ coupled multinomial logistic regressions rather than a single optimization over the global partition function.

#### 4.2.1. Method

We retain the Potts parametrization of fields and couplings introduced in Eq. 17. Similar to mfDCA, sequences are reweighted using $w_m$ defined in Eq. 18 and the effective sample size $M_{\text{eff}}$ defined in Eq. 19.

**Method Setup**

The pivotal idea is to optimize pseudolikelihoods. For site $r$, the conditional distribution given all other sites $\sigma_{\setminus r}$ is

$$P\Big(\sigma_r = A \mid \sigma_{\setminus r}\Big) = \frac{\exp\Big(h_r(A) + \sum_{i\neq r} J_{ri}(A,\sigma_i)\Big)}{\sum_{B=1}^{q}\exp\Big(h_r(B) + \sum_{i\neq r} J_{ri}(B,\sigma_i)\Big)}). \tag{27}$$

The weighted sitewise negative log-pseudolikelihood is

$$g_r(h_r, J_r) = -\frac{1}{M_{\text{eff}}}\sum_{m=1}^{M} w_m \log P\Big(\sigma_r^{(m)} |\ \sigma_{\setminus r}^{(m)}\Big). \tag{28}$$

and the global objective aggregates these points:

$$\mathcal{L}_{\text{pseudo}}(h, J) = \sum_{r=1}^{L} g_r(h_r, J_r). \tag{29}$$

To curtail overfitting, we add convex $l_2$ penalties,

$$R_{\ell_2} = \lambda_h \sum_{r=1}^{L} \|h_r\|_2^2 + \lambda_J \sum_{1 \le i < j \le L} \left\|J_{ij}\right\|_2^2 \tag{30}$$

and minimize

$$\{h^{\mathrm{PLM}}, J^{\mathrm{PLM}}\} = \arg\min_{h,J} \left\{ \ell_{\mathrm{pseudo}}(h, J) + R_{\ell_2}(h, J) \right\} \tag{31}$$

The $\ell_2$ penalty fixes the gauge implicitly by selecting a unique representative among gauge-equivalent parameters. Each $g_r$ is precisely the loss of a multinomial logistic regression (softmax): classes are the $q$ states of $\sigma_r$; features are one-hot encodings of $\{\sigma_i\}_{i \neq r}$ with $(L-1)(q-1)$ degrees of freedom after dropping a reference state. Hence (asymmetric) plmDCA is implemented as $L$ independent, weighted softmax problems with $\mathcal{L}_2$ penalty (e.g. solved with L-BFGS or mini-batch SGD). Two variants are used in practice: asymmetric PLM, which fits each independently and then symmetrizes averaging:

$$\hat{J}_{ij} \leftarrow \frac{1}{2}\left(J_{ij}^{(i)} + J_{ij}^{(j)}\right) \tag{32}$$

and symmetric (joint) PLM, which minimizes $\mathcal{L}_{\mathrm{pseudo}}$ over all parameters at once. Results are typically similar, but optimization in the symmetric variant can be heavier.

**Pair Scoring**

For pair scoring, plmDCA avoids the previously defined Direct Information (DI) as it would introduce a regularization parameter for the pseudocounts, on top of the existing $\lambda_h$ and $\lambda_J$. Instead it considers the Frobenius Norm (FN)

$$\left\|J_{ij}\right\|_2 = \sqrt{\sum_{k,l=1}^{q} J_{ij}(k,l)^2}. \tag{33}$$

Unlike the DI score, the FN is not independent of gauge choice so a direct decision must be made. As noted in [17], the zero-sum gauge minimizes the FN, making it the most appropriate gauge choice available. The procedure is hence (i) convert to zero-sum gauge $J'_{ij}$:

$$J'_{ij}(k,l) = J_{ij}(k,l) - J_{ij}(\cdot,l) - J_{ij}(k,\cdot) + J_{ij}(\cdot,\cdot), \tag{34}$$

where "$\cdot$" denotes a simple average over the $q$ states at that position; (ii) compute the Frobenius norm

$$\mathcal{S}^{\mathrm{FN}}_{ij} = \left\| J'_{ij} \right\|_2 = \sqrt{\sum_{k,l=1}^{q} \left( J'_{ij}(k,l) \right)^2}; \tag{35}$$

(iii) apply Average Product Correction (APC) adjusted from its use in [22] to reduce phylogenetic bias,

$$\mathcal{S}^{\mathrm{CN}}_{ij} = \mathcal{S}^{\mathrm{FN}}_{ij} - \frac{\mathcal{S}^{\mathrm{FN}}_{i\cdot}\mathcal{S}^{\mathrm{FN}}_{\cdot j}}{\mathcal{S}^{\mathrm{FN}}_{\cdot\cdot}}, \tag{36}$$

where $\mathcal{S}^{\mathrm{FN}}_{i\cdot}$ and $\mathcal{S}^{\mathrm{FN}}_{\cdot j}$ are row/column means and $\mathcal{S}^{\mathrm{FN}}_{\cdot\cdot}$ is the grand mean. Residue pairs $(i,j)$ are ranked by $\mathcal{S}^{\mathrm{CN}}_{ij}$ to predict structural contacts.

#### 4.2.2. Limitations

Despite its practical impact, plmDCA remains sensitive to sampling: accurate contact recovery still requires large $M_{\mathrm{eff}}$, and sparse or biased MSAs degrade estimates. Phylogenetic and positional bias persist (reweighting and APC help but do not eliminate them), which can inflate false positives.

#### 4.2.3. Fast plmDCA (2014)

To make plmDCA deployable at scale, two of the original authors and a third collaborator revisited the optimization choices of the method [23]. In this second version, the

asymmetric variant is used: $L$ independent, weighted softmax regressions are computed, one per site, and the final couplings are symmetrized by averaging,

$$J_{ij} = \frac{1}{2}\Big(J_{ij}^{(i)} + J_{ij}^{(j)}\Big). \tag{37}$$

This decomposition reduces per-solve dimensionality and, crucially, enables trivial parallelization across CPU cores or nodes, which is the primary source of the runtime gain. Since each $J_{ij}$ is regularized twice (once in the regression for $i$ and once for $j$), the coupling penalty parameter must be halved relative to the symmetric formulation. Before averaging, the two independently inferred coupling blocks are each shifted into the zero-sum gauge to ensure consistency. For scoring, the method adopts the Corrected Frobenius Norm (CFN): first compute the Frobenius norm of $\hat{J}_{ij}$ excluding the gap state,

$$\mathrm{FN}_{ij} = \sqrt{\sum_{k,l \neq \text{ gap}} \hat{J}_{ij}(k,l)^2}, \tag{38}$$

then apply the Average Product Correction

$$\mathcal{S}_{ij}^{\mathrm{CFN}} = \mathrm{FN}_{ij} - \frac{\mathrm{FN}_{i:}\mathrm{FN}_{:j}}{\mathrm{FN}_{::}} \tag{39}$$

The combination of asymmetric regression and symmetrization yields similar contact accuracy as the original plmDCA, but at a fraction of the runtime, making large protein families and long sequences tractable in practice.

### 4.3. Boltzmann Machine DCA (2018)

In mfDCA accuracy was traded for speed via the small-coupling inversion; in plmDCA, a product of sitewise conditionals was optimized, avoiding the global partition function. bmDCA[24] instead proposes a solution closer to the statistical ideal: fitting the full Potts model by maximizing the true likelihood, using Monte Carlo Markov Chains to estimate intractable expectations. Improving on this, adabmDCA[25] introduces an

adaptive procedure to keep sampling reliable and efficient. The result is a generative model that (by construction) matches the empirical one- and two-site statistics of the reweighted MSA.

#### 4.3.1. Method

bmDCA aims to infer the full Potts model defined as Eq. 17 that reproduces the single- and pair-wise frequencies of the MSA.

The training procedure follows the classical Boltzmann machine learning algorithm:

1. Sampling step. For current parameters {J, h}, estimate model frequencies $p_i, p_{ij}$ by MCMC
2. Update step. Adjust parameters when the estimated frequencies deviate from empirical ones: $\Delta h_i \propto f_i - p_i, \quad \Delta J_{ij} \propto f_{ij} - p_{ij}$
3. Iterate until empirical and model moments match within tolerance.

Regularization is imposed to stabilize the fit and avoid overfitting.

#### 4.3.2. Limitations

- Computational cost: Direct Boltzmann machine learning with MCMC is extremely slow for realistic protein lengths ($10^7 - 10^9$ parameters).
- Sampling difficulty: Accurate moment estimation requires long chains and careful equilibration, making the approach impractical without approximations.
- Scaling: While bmDCA is theoretically optimal (full likelihood optimization), it is limited in practice to small systems where large-scale computation is feasible.

#### 4.3.3. adabmDCA (2021)

adabmDCA revisits the same goal of fitting the full Potts by likelihood maximization, but introduces an adaptive MCMC procedure to make the approach more practical. Instead of relying on fixed sampling parameters, it dynamically tunes equilibration, waiting times, and chain persistance. adabmDCA pushes the bmDCA approach closer to practical usability, at the cost of a more complex implementation and still significant compute requirements.

**Method**

Let the MSA have length L, and M sequences $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_L)$. The maximum-entropy distribution that reproduces chosen moments is the Potts model defined in Eq. 17. To reduce phylogenetic redundancy, assign each sequence a weight $w_m$ as in Eq. 18. Empirical frequencies are reweighted and smoothed with pseudocount $\lambda$ as

$$\begin{gathered} f_i(A) = (1-\lambda) f_i^{\text{data}}(A) + \frac{\lambda}{q}, \quad f_{ij}(A,B) = (1-\lambda) f_{ij}^{\text{data}}(A,B) + \frac{\lambda}{q^2} \\ f_i^{\text{data}}(A) = \frac{1}{M_{\text{eff}}} \sum_m w_m \delta\Big(A, \sigma_i^{(m)}\Big), \quad f_{ij}^{\text{data}}(A,B) = \frac{1}{M_{\text{eff}}} \sum_m w_m \delta\Big(A, \sigma_i^{(m)}\Big)\Big(B, \sigma_j^{(m)}\Big) \end{gathered} \tag{40}$$

A practical starting point for the system is the profile model, an independent-site Potts model where the first empirical moments are matched by means of the fields,

$$h_i^{\text{prof}}(A) = \log f_i(A) + \text{const}, \tag{41}$$

but all couplings are set to zero $J \equiv 0$. In addition, zero or user-provided initial parameters can also work.

**Likelihood and moment matching**

The average log-likelihood of the MSA under $P(\cdot\,|J,h)$ is

$$\mathcal{L}(J,h) = \frac{1}{M} \sum_{m=1}^{M} \left[ \sum_i h_i\Big(\sigma_i^{(m)}\Big) + \sum_{i<j} J_{ij}\Big(\sigma_i^{(m)}, \sigma_j^{(m)}\Big) \right] - \log Z(J,h). \tag{42}$$

As an exponential-family model, $\mathcal{L}$ is concave in the natural parameters, so gradient ascent converges to the unique optimum. The gradients are moment gaps:

$$\frac{\partial \mathcal{L}}{\partial h_i(A)} = f_i(A) - p_i(A), \quad \frac{\partial \mathcal{L}}{\partial J_{ij}(A,B)} = f_{ij}(A,B) - p_{ij}(A,B), \tag{43}$$

where $p_i$ and $p_{ij}$ are model marginals under current $(J,h)$. Hence the update

$$
\begin{aligned}
h_i^{t+1}(A) &\leftarrow h_i^t(A) + \eta_h \Big[ f_i(A) - p_i^{(t)}(A) \Big], \\
J_{ij}^{t+1}(A,B) &\leftarrow J_{ij}^t(A,B) + \eta_J \Big[ f_{ij}(A,B) - p_{ij}^{(t)}(A,B) \Big].
\end{aligned}
\tag{44}
$$

drives the model toward exact moment matching f = p. The obstacle is that $p_i$ and $p_{ij}$ are not analytically computable at scale; adabmDCA estimates them by MCMC at each epoch.

**Estimating model expectations via adaptive MCMC**

At training epoch $t$, run $N_s$ independent Markov chains using Metropolis-Hastings [26], [27] (Gibbs sampling strategy may also be used [28]), each producing $N_c$ samples after an equilibration period $T_{\text{eq}}$ and with an inter-sample waiting time $T_{\text{wait}}$. The Monte Carlo estimators are

$$
\begin{aligned}
p_i^{(t)}(A) &= \frac{1}{N_s N_c} \sum_{m=1}^{N_s N_c} \delta\Big(A, \sigma_i^{(m)}(t)\Big), \\
p_{ij}^{(t)}(A,B) &= \frac{1}{N_s N_c} \sum_{\nu=1}^{N_s N_c} \delta\Big(A, \sigma_i^{(m)}(t)\Big) \delta\Big(B, \sigma_j^{(m)}(t)\Big).
\end{aligned}
\tag{45}
$$

Chains may be transient, reinitialized every epoch, or persistent, initialized only at the first epoch. Equilibration is often sped up through persistence of chains. For more details on the adaptive scheme, refer to Appendix C.

**Convergence and quality control**

A practical convergence proxy is the difference between the empirical and the model two-site connected correlations:

$$
\begin{aligned}
\varepsilon_c &= \max_{i,j,a,B} \Big| c_{ij}^{\text{model}}(A,B) - c_{ij}^{\text{emp}}(A,B) \Big| \quad \text{where} \\
c_{ij}^{\text{model}} &= p_{ij} - p_i p_j, \quad c_{ij}^{\text{emp}} = f_{ij} - f_i f_j
\end{aligned}
\tag{46}
$$

with a target $\varepsilon_c \approx 10^{-}2$ In addition to this, some other commonly used diagnostics is the Pearson correlation between $c^{\text{model}}$ and $c^{\text{emp}}$, one- and two-site fitting errors, and optionally, a third connected correlation on a subset of triples is used to assess generative fidelity beyond pairwise constraints.

### Priors and sparsity

A fully connected Potts model has $\sim \frac{L(L-1)}{2}q^2 + Lq$, meaning a number in the order of $10^7$-$10^9$ parameters for a realistic $L$ [25]. Due to the finite sample size of MSAs, they rarely contain enough independent information to estimate all of them robustly. Not controlling this uncertainty could lead to overfitting, high variance or instability, and bad conditioning in the model. To address these issues, adabmDCA employs two complementary strategies. First, we can place a prior on $P(J, h)$ and maximize the posterior, equivalent to adding penalties to the objective. The two standard choices are the $\ell_1$ and $\ell_2$ priors:

$$\begin{aligned} R_{\ell_1}(J, h) &= \theta_{1,h} \sum_i \|h_i\|_1 + \theta_{1,J} \sum_{i<j} \|J_{ij}\|_1 \\ R_{\ell_2}(J, h) &= \theta_{2,h} \sum_i \|h_i\|_2^2 + \theta_{2,J} \sum_{i<j} \|J_{ij}\|_2^2 \end{aligned} \tag{47}$$

Under $\ell_2$, the gradients are shrunk toward zero:

$$\begin{aligned} \frac{\partial}{\partial h_i(A)} &: f_i(A) - p_i(A) - \theta_{2,h} h_i(A), \\ \frac{\partial}{\partial J_{ij}(A, B)} &: f_{ij}(A, B) - p_{ij}(A, B) - \theta_{2,h} J_{ij}(A, B). \end{aligned} \tag{48}$$

Under $\ell_1$, they include subgradient terms that promote exact zeros.

$$\begin{aligned}\frac{\partial}{\partial h_i(A)} &: f_i(A) - p_i(A) - \theta_{1,h}\ \mathrm{sign}(h_i(A)),\\ \frac{\partial}{\partial J_{ij}(A,B)} &: f_{ij}(A,B) - p_{ij}(A,B) - \theta_{1,h}\ \mathrm{sign}\big(J_{ij}(A,B)\big).\end{aligned} \tag{49}$$

$\ell_2$ reduces variance and improves conditioning by smoothly shrinking all parameters. It selects a unique gauge and tends to preserve the relative ordering of strong couplings. On the other hand, $\ell_1$ induces sparsity by zeroing weak parameters, also reducing overfitting, though at a cost of biasing small effects downward. Generally, in stochastic settings, elastic-net is used (a combination of both parameters), that stabilizes training near zero. In addition, a separate parameter is used for fields and couplings, typically regularizing $J$ more strongly than $h$.

The second method is introducing sparsity via pruning or decimation. The reason for this is that true contact maps in nature are indeed sparse; most residue pairs are not in direct physical contact. Encoding this structural prior can reduces variance and speed up learning. There are two approaches:

1. A priori topology. Reduce the number of parameters by starting from a restricted edge set, for example pairs with high MI, and learn only those $J_{ij}$, omitting the rest.
2. Information-based decimation. In this approach, start dense and iteratively remove the least informative couplings until target sparsity. This can be done by comparing the KL divergence for a candidate element. This directly controls overfitting by only keeping the parameters that actually affect the model's predictions.

In short, pruning and decimation prevent overfitting by removing parameters that don't materially alter the model, and have the added benefit of aligning with the biological prior of contact sparsity.

### 4.4. Autoregressive Network DCA

Building on the pseudolikelihood maximization of plmDCA, and Boltzmann-machine parametrization of bmDCA, arDCA was introduced as a faster and more efficient

alternative. While plmDCA is limited in scope and bmDCA suffers from slow MCMC sampling, reformulating the problem as an autoregressive network provides some key improvements. The key idea is to decompose the joint sequence probability distribution its conditionals, thereby turning it into a supervised learning task. In this way, arDCA emerges as a powerful model for extracting structural and functional information from the rapidly growing protein databases.

#### 4.4.1. Method

In arDCA, the exponential-family MaxEnt distribution underlying previous DCA methods is factorized into conditional probabilities, predicting each residue from its predecessors. This follows directly from the chain rule of probability:

$$P(\boldsymbol{\sigma}) = \prod_{i=1}^{L} P(\sigma_i | \sigma_1, ..., \sigma_{i-1}) \tag{50}$$

Inspired by approaches in classical [29] and quantum [30] statistical mechanics, the conditional distribution is parametrized as:

$$P(a_i \mid a_{i-1}, ..., a_1) = \frac{\exp\Big(h_i(a_i) + \sum_{j=1}^{i-1} J_{ij}(a_i, a_j)\Big)}{\sum_{a_i} \exp\Big(h_i(a_i) + \sum_{j=1}^{i-1} J_{ij}(a_i, a_j)\Big)} \tag{51}$$

This is a multiclass softmax regression [31], generalizing logistic regression to multiple residue states. The model is deinfed by site-specific fields $h_i(a)$ and directed couplings $J_{ij}(a, b)$, and was therefore termed arDCA. For comparison, the authors also introduced a simpler profile model (independent-site model), which only includes field terms. Its joint probability factorizes across sites,

$$P(a_1, ..., a_L) = \prod_{i=1...L} f_i(a_i) \tag{52}$$

. making it a useful baseline.

Although arDCA shares the same number of parameters as standard DCA, their interpretation differs. Standard DCA has symmetric couplings, $J_{ij}(a,b) = J_{ji}(b,a)$, whereas Eq. 51 couplings are directed, describing only the influence of site $j$ on $i$ for $j < i$. Thus, only the lower-triangular part of the coupling matrix is populated. Inference in arDCA, as in plmDCA, is performed via pseudo-likelihood maximization[32], which allows exact gradient computation from data–unlike bmDCA, which requires costly MCMC sampling. A key difference is that plmDCA conditions each residue on all others in the sequence, often symmetrizing the resulting couplings to align with Potts models. This symmetrization shifts parameters away from their maximum-likelihood values, reducing the model's generative accuracy. By contrast, arDCA does not require symmetrization, preserving likelihood consistency.

The chain rule decomposition is valid for any site ordering, but once the conditional parametrization of Eq. 51 is introduced, the likelihood becomes order dependent. Optimizing over $L!$ permutations is infeasible in practice, so the original authors proposed entropic ordering as a principled heuristic, sites are ranked from lowest entropy (most conserved position) to the highest entropy (most variable). This choice reflects a natural interpretation: conserved positions provide little additional information if occupied by their dominant residue, whereas deviations must be compensated downstream by more variable sites. By placing these variable positions later in the order, the model can capture compensatory effects in a structured way. Although we do not optimize or reorder sites in our implementation, entropic ordering provides useful context for how positional dependencies can be learned by arDCA.

**Parameter inference**

The inference of the parameters is done through likelihood maximization. Following a Bayesian setting with a uniform prior, the optimal parameters are those that maximize the probability of the data[4]:

[4]Here $a$ is used in place of A for sake of space

$$
\begin{aligned}
\{J^*, h^*\} &= \arg\max_{\{J,h\}} P(\mathcal{M}|\{J,h\}) \\
&= \arg\max_{\{J,h\}} \log P(\mathcal{M}|\{J,h\}) \\
&= \arg\max_{\{J,h\}} \sum_{m=1}^{M} \log \prod_{i=1}^{L} P(a_i^m | a_{i-1}^m, ..., a_1^m) \\
&= \arg\max_{\{J,h\}} \sum_{m=1}^{M} \sum_{i=1}^{L} \log P(a_i^m | a_{i-1}^m, ..., a_1^m)
\end{aligned}
\tag{53}
$$

Each $h_i(a)$ and $J_{ij}(a,b)$ is present in only one conditional probability $P(a_i|a_{i-1},...,a_1)$, thus we can maximize each conditional probability independently in Eq. 53:

$$
\left\{J_{ij}^*, h_i^*\right\} = \arg\max_{\{J_{ij},h_i\}} \sum_{m=1}^{M} \left[ h_i(a_i^m) + \sum_{j=1}^{i-1} J_{ij}\left(a_i^m, a_j^m\right) - \log z_i(a_{i-1}^m, ..., a_1^m) \right] \tag{54}
$$

where

$$
z_i(a_{i-1}, .., a_1) = \sum_{a_i} \exp\left( h_i(a_i) + \sum_{j=1}^{i-1} J_{ij}(a_i, a_j) \right) \tag{55}
$$

is the normalization factor of the conditional probability of $a_i$.

Taking the derivative with respect to $h_{i(a)}$ or $J_{ij}(a,b)$, with $j = 1, ..., i-1$, we get:

$$
\begin{aligned}
0 &= \frac{1}{M} \sum_{m=1}^{M} \left[ \delta(a, a_i^m) - \frac{\partial \log z_i(a_{i-1}^m, .., a_1^m)}{\partial h_i(a)} \right] \\
0 &= \frac{1}{M} \sum_{m=1}^{M} \left[ \delta(a, a_i^m)\delta\left(b, a_j^m\right) - \frac{\partial \log z_i(a_{i-1}^m, .., a_1^m)}{\partial J_{ij}(a,b)} \right].
\end{aligned}
\tag{56}
$$

Using Eq. 55, we find

$$\begin{aligned}\frac{\partial \log z_i(a_{i-1}^m, .., a_1^m)}{\partial h_i(a)} &= P(a_i = a \mid a_{i-1}^m, ..., a_1^m) \\ \frac{\partial \log z_i(a_{i-1}^m, .., a_1^m)}{\partial J_{ij}(a,b)} &= P(a_i = a \mid a_{i-1}^m, ..., a_1^m)\delta\Big(a_j^m, b\Big).\end{aligned} \tag{57}$$

The set of equations reduces to a simple form:

$$\begin{aligned}f_i(a) &= \langle P(a_i = a \mid a_{i-1}^m, ..., a_1^m)\rangle_{\mathcal{M}}, \\ f_{ij}(a,b) &= \langle P(a_i = a \mid a_{i-1}^m, ..., a_1^m)\delta\Big(a_j^m, b\Big)\rangle_{\mathcal{M}},\end{aligned} \tag{58}$$

where $\langle\cdot\rangle_{\mathcal{M}} = \frac{1}{M}\sum_{m=1}^{M} \cdot^m$ denotes the empirical average. The first variable, $i = 1$, is unconditioned, therefore the coupling equation is $J \equiv 0$ and the equation for the field is the profile model also used in bmDCA, $h_1(a) = \log f_1(a) + \text{const}$.

Unlike bmDCA, the equations do not enforce exact matching between model marginals and empirical frequencies. The ability to reproduce the frequencies is thus a good proxy for the generative properties of the model, on top of the fitting quality of current parameters. In practice, the parameters are updated using gradient descent on the likelihood. These gradients are exact as we can take the expectations directly over the MSA. For regularization, arDCA makes use of $\ell_2$ penalties, with different strengths depending on whether the task is generating sequences or just contact prediction. It was noted that small regularization values improved the generative quality while larger were beneficial for contact prediction.

#### 4.4.2. Key Contributions

arDCA represents a major advance over previous DCA methods by enabling exact gradient computations from the data. This eliminates the need for costly MCMC sampling, leading to training speeds that are two to three orders of magnitude faster while maintaining, or even improving, predictive accuracy[33]. The resulting model is therefore lightweight and computationally scalable, making it feasible to apply to large

protein families, although DeepSequence[34] prevailed in prediction accuracy of these families.

A second key contribution is that arDCA allows for the exact calculation of sequence probabilities, a task intractable for previous models. In bmDCA, only unnormalized sequence weights could be computed, with the partition function requiring expensive thermodynamic integration. By contrast, each conditional probability in arDCA is normalized locally. This reduces the computational burden from summing over $q^L$ possible sequences to only $L$ sums over q residue states. This efficiency enables direct sequence-level comparisons across models, which is especially valuable for applications such as homology detection, protein family classification, and model-based sequence evaluation.

## 5. Implementation and Exploration

The original ArDCA model was developed in Julia, a high-performance numerical and scientific computing language. Julia's strengths lie in just-in-time (JIT) compilation, seamless handling of linear algebra, and built-in support for parallelism, making it well-suited for implementing large-scale statistical models. In contrast, the re-implementation was carried out in Python, a widely adopted language for machine learning and data science. While Python itself is generally slower due to its interpreted nature, performance-critical operations are offloaded to highly optimized libraries (e.g., NumPy, SciPy, PyTorch), which use underlying C/C++ or Fortran code.

### Important Differences Between Julia and Python

There are structural differences between Julia and Python that make porting between them not a simple one-to-one translation. To begin, Julia uses column-major order, prioritizing very fast and easily cachable operations for the columns. Instead, Python is structured with row-major order where row-wise operations are more optimized. The impact of this difference is that loops and reshaping operations defined in Julia need to

be reconsidered to have the same speed and memory efficiency when defined in Python. Another difference is that Julia is 1-indexed whereas Python is 0-indexed. This has no effect on the performance, but it changes the bounds on loops, the indexing of objects, and the range functions.

Furthermore, loops have different behaviors in the languages. Julia loops are compiled down to efficient machine code with JIT. The `inbounds` macro removes the bounds checking on the loops, making them fast without the need for vectorization. On the other hand, pure Python loops are slow. To reach the same level of efficiency, vectorization is needed in Python, implemented via NumPy or broadcasting.

The final important difference is in compilation versus interpretation. Julia JIT compiles functions to machine code, which has an initial compilation cost, but allows future calls to run much faster. On the other hand, Python is interpreted, thus heavy computations require external libraries which are precompiled in other, more efficient, languages.

## 5.1. Implementation details

We implement the autoregressive network within the PyTorch ecosystem, using the `nn.Module` interface to leverage its training and optimization capabilities. The central operation of the model is the computation of the autoregressive logits, which define the conditional distribution at each sequence position.

### Logit Computation

For site $i$, the logit vector is given by

$$z[m, i, a] = h[i, a] + \sum_{j<i} \sum_{b} J[i, j, a, b] X[m, j, b] \tag{59}$$

where $h$ and $J$ are the local biases and pairwise couplings, and $X$ is a tensor holding the one-hot encoding of the symbol $b$ observed at site $j$ in sequence $m$.

To compute the autoregressive logits, our initial implementation employed a masked matrix multiplication, performed in a single step using the `einsum` operator. While correct, and more efficient than naive loops, calculating the full interaction matrix is computationally inefficient because the upper-triangular entries are all masked to zero. To address this, we devised a more efficient formulation that exploits the block-sparse structure of the coupling matrix. Specifically, we collect all lower-triangular index pairs $(i, j)$ with $j < i$, extract the corresponding coupling blocks $J[i, j]$, and accumulate their contributions through an `einsum` operation. The resulting implementation only computes terms required by the autoregressive factorization, removing unnecessary computation.

```
def compute_ar_logits(self, X_oh: torch.Tensor):
        """z[m,i,a] = h[i,a] + sum_{j<i} sum_b J[i,j,a,b] * X[m,j,b]"""
        X_oh = X_oh.to(self.J.dtype)
        M, L, q = X_oh.shape
        logits = self.h_pos.unsqueeze(0).expand(M, -1, -1).clone()  # (M,L,q)

        # collect lower-triangular index pairs
        i_idx, j_idx = torch.tril_indices(L, L, offset=-1)
        J_blocks = self.J[i_idx, j_idx]   # (n_pairs, q, q)
        X_blocks = X_oh[:, j_idx]         # (M, n_pairs, q)

        # compute pairwaise contributions and accumulate into logits
        contrib = torch.einsum("mpq,pqr->mpr", X_blocks, J_blocks)
        logits = logits.index_add(1, i_idx, contrib)
        return logits
```

With this formulation, the computational complexity scales with the number of lower-triangular pairs rather than the full $L \times L$ coupling matrix. For long sequences, this is especially beneficial. In addition, by avoiding the construction of the full masked interaction matrix, the memory footprint is roughly halved, from $L^2$ couplings to $\frac{L(L-1)}{2}$.

Throughout the model, an explicit binary mask `J_mask` is maintained that encodes the lower-triangular structure, to ensure consistency in the autoregressive aspect. After each optimization step, unused entries of $J$ are clamped to zero.

**Training and Evaluation details**

The model is initialized with the local bias of the first position, $h[0]$, estimated from the empirical symbol frequencies observed at site 0. The rest of the parameters are drawn from small random distributions. Training incorporates sequence reweighting to correct for redundancy in MSAs. The weights for each sequence are computed at the start, with parameter `theta` controlling the sequence similarity threshold, and the effective sample size is defined as in Eq. 19. The model can be optimized with either L-BFGS, suited for energy-based models, or AdamW, offering a more scalable and robust option for large datasets. Regularization is applied separately to the field and couplings via $\ell_2$ -norm penalties with hyperparameters $\lambda_h$ and $\lambda_J$.

For evaluation, the model reports the negative log-likelihood (NLL) per position,

$$\mathrm{nll}_{m,i} = -w_m \log p_\theta\big(X_{m,i} | X_{m,<i}\big) \tag{60}$$

with the average NLL per position

$$\overline{\mathrm{NLL}} = \frac{1}{ML} \sum_{m=1}^{M} \sum_{i=1}^{L} \mathrm{nll}_{m,i}. \tag{61}$$

Another metric used for evaluation is the perplexity,

$$\text{Perplexity} = \exp\Big(\overline{\mathrm{NLL}}\Big). \tag{62}$$

It is widely used in Natural Language Processing and represents the average number of equally likely choices the model is making per residue[35]. Lower perplexity means the model is more confident and better fits the data. The effective sample size is also

reported, to make the results more easily comparable across different datasets. Finally, an ancestral sampling procedure is provided, which generates new sequences position by position using the learned conditional distributions. This procedure is inherently slower than parallel inference, but it guarantees that samples are consistent with the autoregressive structure.

## 5.2. Experiments

### 5.2.1. Effect of Gaps in Multiple Sequence Alignments

When constructing MSAs, gaps are intro introduced to better align homologous positions across sequences. These gaps serve two main purposes: (i) ensuring residues align correctly at homologous sites, and (ii) extending sequences to a specific length for models like ArDCA, that require fixed length input.

Since the presence of gaps can influence the statistical properties of the alignment, we examined how restricting the fraction of gaps affects results. Specifically, we introduced two filtering parameters:

- `max_gap_fraction`: sets the maximum allowed fraction of gaps per sequence.
- `max_col_gap_fraction`: sets the maximum allowed fraction of gaps in a column.

For the first experiment, we varied `max_gap_fraction`, while holding the other fixed, with three thresholds: $0.05, 0.1,$ and $1$. 1 does not exclude any sequences, therefore is used as a baseline for comparison. The difference in time of the models was also recorded, but given the small nature of the change, no conclusive results were observed.

| `max_gap_fraction` | M | $M_{\text{eff}}$ | Perplexity |
|---|---|---|---|
| 0.05 (-6%) | 12784 | 3876.5 | 1.72 |
| 0.1 (-1.6%) | 13376 | 4145.4 | 1.63 |
| 1 | 13600 | 4248.8 | 1.77 |

Table 1: Results for `max_gap_fraction` experiment

Restricting the allowed fraction of gaps resulted in the removal of sequences and a corresponding decrease in the effective number of sequences $M_{\text{eff}}$. The strictest filterint criterion (0.03) removed approximately 8.7% of sequences relative to the baseline, yielding a smaller dataset and slightly worse perplexity. The lowest perplexity (1.63) and fastest time ($344s$) was observed when 10% of gaps were excluded. Although the full dataset had a higher $M_{\text{eff}}$, the higher perplexity suggests that allowing too many gaps may negatively impact the quality fo the alignment.

A similar exploration was performed for the column gaps with values $0.025, 0.1, 1$. Again, 1 represents the baseline with no columns removed.

| `max_col_gap_fraction` | L | $M_{\text{eff}}$ | Perplexity |
|---|---|---|---|
| 0.025 (-7.5%) | 49 | 4363.0 | 1.79 |
| 0.1 (-3.7%) | 51 | 4105.9 | 1.62 |
| 1 | 53 | 4248.8 | 1.77 |

Table 2: Results for `max_col_gap_fraction` experiment

Both sequence-level and site-level filtering yield a better result when using moderate thresholds, where exclude just the data with the most gaps improves model performance. Overly aggressive filtering discards useful information and hurts perplexity, while using the full dataset also doesn't reach the best performance.

### 5.2.2. Datasets

Different protein families were analyzed and explored in the evaluation of the model.

| **Protein Family** | M | L | $M_{\text{eff}}$[5] |
|---|---|---|---|
| PF00014 | 13600 | 53 | 4208.5 |
| PF00076 | 137605 | 70 | |
| PF00072 | 823798 | 112 | |
| PF13354 | 7515 | 202 | |

[5]Calculated with sequence identity threshold 0.8

# 6. Results and Discussion

## 6.1. Training Behavior

During training of the model, we implemented an optimization loop with AdamW optimize. The model parameters were initialized on the training set and updated iteratively by computing exact gradients of the negative log-likelihood (NLL). A validation fraction (0.1) of the data was held out, and both training and validation losses were tracked throughout training. To prevent overfitting, early stopping was employed with a patience of 20 evaluation intervals, halting training when no further improvement in validation loss was observed. The training process produced smooth convergence curves, reflecting the stability of the exact gradient computations in ArDCA. As illustrated in Figure Figure 2, the training NLL decreased steadily over 200 epochs across all tested versions of the model, from an initial value above 160 to below 80. Importantly, the validation loss exhibited the same monotonic decrease as the training loss, indicating minimal overfitting and robust generalization. The observed behavior demonstrates that ArDCA benefits from direct gradient-based optimization, yielding stable convergence and consistent performance across different runs.

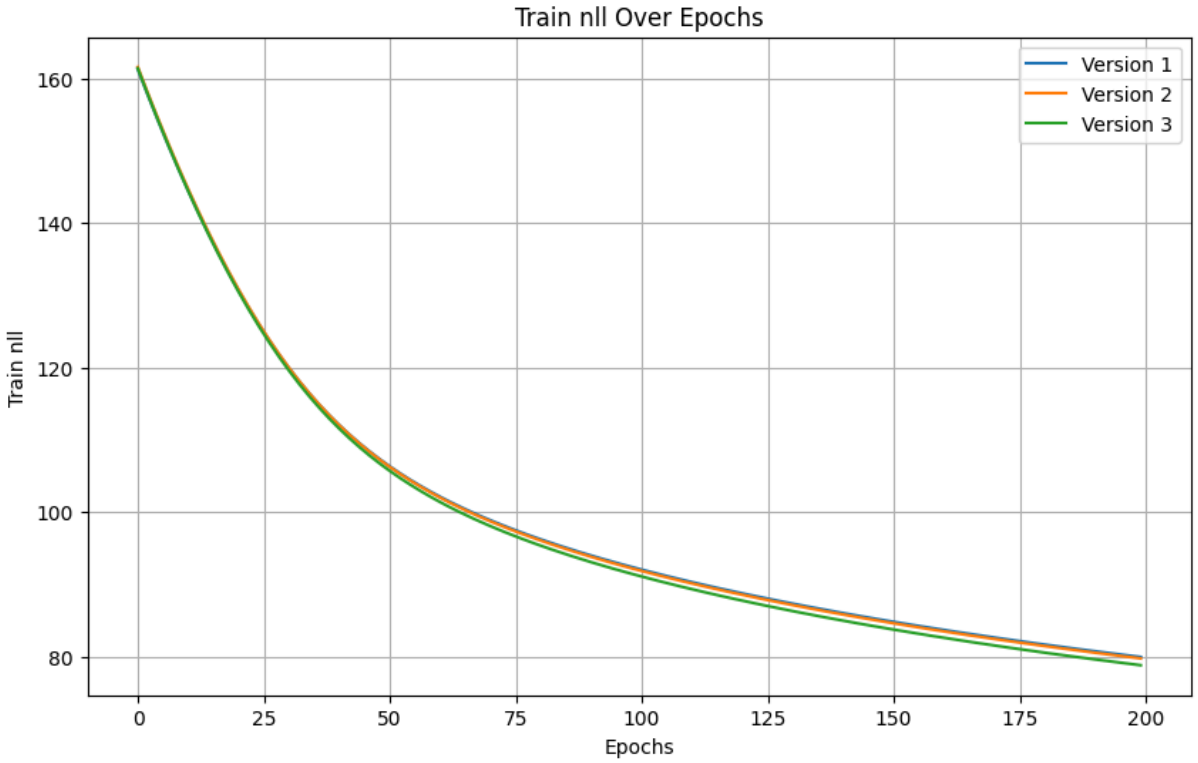


Figure 2: Training NLL Plotted for `max_gap_fraction` experiment

### 6.2. Generative Model Quality

The final test for the models is their ability to generate accurate sequences. For this we can use AlphaFold's plDDT [36], https://colab.research.google.com/github/sokrypton/ColabFold/blob/main/AlphaFold2.ipynb

### 6.3. Limitations and Insights

The Python implementation, while optimized through a lower-triangular masked multiplication, remains less efficient than the original Julia version due to the differences in performance between the languages. Beyond implementation, the architecture itself presents inherent limitations. It is restricted to fixed length sequences, limiting its flexibility in modeling families with variable domain lengths. Moreover, its performance is strongly tied to the quality of the multiple sequence alignment. Since the model learns from patterns of co-variation in the alignment, families with limited diversity or strong phylogenetic bias may provide insufficient or misleading signals. In such cases, the model risks poor generalization, generating sequences that fail to capture the true structural and functional constraints of the protein family.

## 7. Conclusion

In this work, I reimplemented the autoregressive DCA model in Python and demonstrated its viability as a practical tool for protein sequence analysis. The PyTorch-based implementation introduced an efficient block-sparse formulation for computing autoregressive logits, reducing the computational cost and memory usage. Empirical validation across several protein families confirmed the correctness and robustness of the model, with smooth training dynamics and consistent improvements in negative log-likelihood, perplexity, and structural plausibility as measured by pLDDT. Additional experiments on gap filtering highlighted the importance of sequence quality control, while tests on larger Pfam families showed that the method remains tractable at scale. In the future, this work could benefit from GPU acceleration, to improve its speed, but also allow it to more easily work with large datasets. Overall, this demonstrates that ArDCA can be adapted beyond its Julia origin and remain a powerful protein-generating model, allowing for downstream tasks like contact prediction or structural modeling.

## Bibliography


[1] M. Weigt, "Coevolutionary Analysis of Protein-Protein Interactions." [Online]. Available: https://www.youtube.com/watch?v=IYA8WEsUcG0

[2] J. Jumper *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021, doi: 10.1038/s41586-021-03819-2.

[3] "The Shape and Structure of Proteins," in *Molecular Biology of the Cell*, 4th ed., New York: Garland Science, 2002. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK26830/

[4] European Bioinformatics Institute (EMBL-EBI), "What are protein families?." [Online]. Available: https://www.ebi.ac.uk/training/online/courses/protein-classification-intro-ebi-resources/protein-classification/what-are-protein-families/

[5] E.-E. Training, "What is phylogenetics?." [Online]. Available: https://www.ebi.ac.uk/training/online/courses/introduction-to-phylogenetics/what-is-phylogenetics/

[6] M. Wiltgen, "Algorithms for Structure Comparison and Analysis: Homology Modelling of Proteins," *Encyclopedia of Bioinformatics and Computational Biology*. Academic Press, pp. 38–61, 2019. doi: https://doi.org/10.1016/B978-0-12-809633-8.20484-6.

[7] moshi4, "pyMSAviz: MSA (Multiple Sequence Alignment) visualization Python package." 2024.

[8] C. Zhang *et al.*, "The Historical Evolution and Significance of Multiple Sequence Alignment in Molecular Structure and Function Prediction," *Biomolecules*, vol. 14, no. 12, 2024, doi: 10.3390/biom14121531.

[9] B. Adhikari and J. Cheng, "Protein Residue Contacts and Prediction Methods," *Methods in Molecular Biology*, vol. 1415, pp. 463–476, 2016, doi: 10.1007/978-1-4939-3572-7_27.

[10] N. Dietler, U. Lupo, and A.-F. Bitbol, "Impact of phylogeny on structural contact inference from protein sequence data," *Journal of the Royal Society, Interface*, vol. 20, no. 199, 2023, doi: 10.1098/rsif.2022.0707.

[11] E. T. Jaynes, "Information Theory and Statistical Mechanics," *Physical Review*, vol. 106, no. 4, pp. 620–630, May 1957, doi: 10.1103/PhysRev.106.620.

[12] E. T. Jaynes, "Information Theory and Statistical Mechanics. II," *Physical Review*, vol. 108, no. 2, pp. 171–190, 1957, doi: 10.1103/PhysRev.108.171.

[13] J. Paul Penfield, "Lecture Notes for Information and Entropy." 2003.

[14] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423623–656, 1948.

[15] E. Ising, "Beitrag zur Theorie des Ferromagnetismus," *Zeitschrift für Physik*, vol. 31, no. 1, pp. 253–258, 1925, doi: 10.1007/BF02980577.

[16] R. B. Potts, "Some generalized order-disorder transformations," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 48, no. 1, pp. 106–109, 1952, doi: 10.1017/S0305004100027419.

[17] S. H. H. J. H. T. Weigt M White RA, "Identification of direct residue contacts in protein-protein interaction by message passing," *Proc Natl Acad Sci USA*, vol. 106, pp. 67–72, 2009, doi: 10.1073/pnas.0805923106.

[18] F. Morcos *et al.*, "Direct-coupling analysis of residue coevolution captures native contacts across many protein families," *Proceedings of the National Academy of Sciences*, vol. 108, no. 49, pp. E1293–E1301, 2011, doi: 10.1073/pnas.1111471108.

[19] T. Plefka, "Convergence condition of the TAP equation for the infinite-ranged Ising spin glass model," *Journal of Physics A: Mathematical and General*, vol. 15, no. 6, pp. 1971–1978, 1982, doi: 10.1088/0305-4470/15/6/035.

[20] A. Georges and J. S. Yedidia, "How to expand around mean-field theory using high-temperature expansions," *Journal of Physics A: Mathematical and General*, vol. 24, no. 9, pp. 2173–2192, 1991, doi: 10.1088/0305-4470/24/9/018.

[21] A. Haldane and R. M. Levy, "Influence of multiple-sequence-alignment depth on Potts statistical models of protein covariation," *Physical Review E*, vol. 99, no. 3, Mar. 2019, doi: 10.1103/physreve.99.032405.

[22] D. T. Jones, D. W. A. Buchan, D. Cozzetto, and M. Pontil, "PSICOV: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments," *Bioinformatics*, vol. 28, no. 2, pp. 184–190, 2012, doi: 10.1093/bioinformatics/btr638.

[23] M. Ekeberg, T. Hartonen, and E. Aurell, "Fast pseudolikelihood maximization for direct-coupling analysis of protein structure from many homologous amino-acid sequences," *Journal of Computational Physics*, vol. 276, pp. 341–356, 2014, doi: https://doi.org/10.1016/j.jcp.2014.07.024.

[24] M. Figliuzzi, P. Barrat-Charlaix, and M. Weigt, "How Pairwise Coevolutionary Models Capture the Collective Residue Variability in Proteins," *Molecular Biology and Evolution*, vol. 35, no. 4, pp. 1018–1027, Apr. 2018, doi: 10.1093/molbev/msy007.

[25] A. P. Muntoni, A. Pagnani, M. Weigt, and F. Zamponi, "adabmDCA: adaptive Boltzmann machine learning for biological sequences," *BMC Bioinformatics*, vol. 22, no. 528, 2021, doi: 10.1186/s12859-021-04441-9.

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953, doi: 10.1063/1.1699114.

[27] W. K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970, [Online]. Available: http://www.jstor.org/stable/2334940

[28] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 721–741, 1984, doi: 10.1109/TPAMI.1984.4767596.

[29] D. Wu, L. Wang, and P. Zhang, "Solving statistical mechanics using variational autoregressive networks," *Physical Review Letters*, vol. 122, no. 8, p. 80602, 2019, doi: 10.1103/PhysRevLett.122.080602.

[30] O. Sharir, Y. Levine, N. Wies, G. Carleo, and A. Shashua, "Deep autoregressive models for the efficient variational simulation of many-body quantum systems," *Physical Review Letters*, vol. 124, no. 2, p. 20503, 2020, doi: 10.1103/PhysRevLett.124.020503.

[31] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York: Springer Science & Business Media, 2009.

[32] S. Balakrishnan, H. Kamisetty, J. G. Carbonell, S.-I. Lee, and C. J. Langmead, "Learning generative models for protein fold families," *Proteins: Structure, Func-*

*tion, and Bioinformatics*, vol. 79, no. 4, pp. 1061–1078, 2011, doi: 10.1002/prot.22934.

[33] J. Trinquier, G. Uguzzoni, A. Pagnani, F. Zamponi, and M. Weigt, "Efficient generative modeling of protein sequences using simple autoregressive models," *Nature Communications*, vol. 12, no. 1, p. 5800, 2021, doi: 10.1038/s41467-021-25756-4.

[34] A. J. Riesselman, J. B. Ingraham, and D. S. Marks, "Deep generative models of genetic variation capture the effects of mutations," *Nature Methods*, vol. 15, no. 10, pp. 816–822, 2018.

[35] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn, "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling," *CoRR*, 2013, [Online]. Available: http://arxiv.org/abs/1312.3005

[36] E.-E. Training, "pLDDT: Understanding local confidence." [Online]. Available: https://www.ebi.ac.uk/training/online/courses/alphafold/inputs-and-outputs/evaluating-alphafolds-predicted-structures-using-confidence-scores/plddt-understanding-local-confidence/

## A Full Langrage Multipliers Calculation for Maximum Entropy Principle

$$
\begin{aligned}
p_i &= e^{-\lambda-\mu f(x_i)} \\
\sum_i p_i &= 1 \\
\sum_i e^{-\lambda-\mu f(x_i)} &= 1, \text{ factor out } \ e^{-\lambda} \\
e^{-\lambda} \sum_i e^{-\mu f(x_i)} &= 1,
\end{aligned}
\tag{63}
$$

Define the partition function $Z(\mu)$ :

$$
\begin{aligned}
Z(\mu) &= \sum_i e^{-\mu f(x_i)}, \text{ thus} \\
e^{-\lambda} Z(\mu) &= 1 \Rightarrow \lambda = \ln Z(\mu)
\end{aligned}
\tag{64}
$$

## B Small Coupling Mean-Field Derivation

Start with the perturbed Hamiltonian

$$\mathcal{H}(\alpha) = -\alpha \sum_{1 \le i < j \le L} e_{ij}(A_i, A_j) - \sum_{i=1}^{L} h_i(A_i) \tag{1}$$

which allows interpolation between independent couplings, $\alpha = 0$, and the original model, $\alpha = 1$. We also define the Gibbs potential

$$-\mathcal{G}(\alpha) = \ln \left[ \sum_{\{A_i | i=1,\dots,L\}} e^{-\mathcal{H}(\alpha)} \right] - \sum_{i=1}^{L} \sum_{B=1}^{q-1} h_i(B) P_i(B) \tag{2}$$

as the Legendre transform of the free energy $\mathcal{F} = -\ln Z$. The fields can be found via

$$h_i(A) = \frac{\partial \mathcal{G}(\alpha)}{\partial P_i(A)}, \tag{3}$$

and

$$\left(C^{-1}\right)_{ij}(A, B) = \frac{\partial h_i(A)}{\partial P_j(B)} = \frac{\partial^2 \mathcal{G}(\alpha)}{\partial P_i(A) \partial P_j(B)}. \tag{4}$$

Our aim is to expand the Gibbs potential up to first order around the independent-site case $\alpha = 0$,

$$\mathcal{G}(\alpha) = \mathcal{G}(0) + \left. \frac{d\mathcal{G}(\alpha)}{d\alpha} \right|_{\alpha=0} \alpha + \mathcal{O}(\alpha^2) \tag{5}$$

**Independent-site approximation**

To start, let us consider the Gibbs potential in $\alpha = 0$. In this case, the Gibbs potential equals the negative entropy of an ensemble of $L$ uncoupled Potts spins,

$$\begin{aligned}\mathcal{G}(0) &= \sum_{i=1}^{L}\sum_{A=1}^{q} P_i(A)\ln P_i(a) \\ &= \sum_{i=1}^{L}\sum_{A=1}^{q-1} P_i(A)\ln P_i(a) + \sum_{i=1}^{L}\left[1-\sum_{A=1}^{q-1}P_i(A)\right]\ln\left[1-\sum_{A=1}^{q-1}P_i(A)\right]\end{aligned} \tag{6}$$

**Mean-field approximation**

To get the first order in Eq. 5, we have to determine the derivative at $\alpha = 0$. Recalling the definition of Gibbs potential in Eq. 2,

$$\begin{aligned}\frac{d\mathcal{G}(\alpha)}{d\alpha} &= \frac{-d}{d\alpha}\ln Z(\alpha) - \sum_{i=1}^{L}\sum_{A=1}^{q-1}\frac{dh_i(A)}{d\alpha}P_i(A) \\ &= -\sum_{\{A_i\}}\left[\sum_{i<j}e_{ij}(A_i,A_j)+\sum_i\frac{dh_i(A)}{d\alpha}\right]\frac{e^{-\mathcal{H}(\alpha)}}{Z(\alpha)} - \sum_{i=1}^{L}\sum_{A=1}^{q-1}\frac{dh_i(A)}{d\alpha}P_i(A) \\ &= -\left\langle\sum_{i<j}e_{ij}(A_i,A_j)\right\rangle_\alpha\end{aligned} \tag{7}$$

The first derivative of the Gibbs potential with respect to $\alpha$ this is the average of the coupling term in the Hamiltonian. At $\alpha = 0$, this average can be done easily due to the joint distribution of all variables becoming factorized over the single sites,

$$\left.\frac{d\mathcal{G}(\alpha)}{d\alpha}\right|_{\alpha=0} = -\sum_{i<j}\sum_{A,B}e_{ij}(A_i,A_j)P_i(A)P_j(B). \tag{8}$$

Plugging this and Eq. 6 into Eq. 5, we find the first-order approximation of the Gibbs potential. The first and second partial derivatives with respect to $P_i(A)$ provide self-consistent equations for the local fields,

$$\frac{P_i(A)}{P_i(q)} = \exp\left(h_i(A) + \sum_{\{j|j\neq i\}}\sum_{B=1}^{q-1}e_{ij}(A,B)P_j(B)\right) \tag{9}$$

and the inverse of the connected correlation matrix,

$$\left(C^{-1}\right)_{ij}(A,B)\Big|_{\alpha=0} = \begin{cases} -e_{ij}(A,B) \text{ for } i \neq j \\ \frac{\delta(A,B)}{P_i(A)} + \frac{1}{P_i(q)} \text{ for } i = j \end{cases} . \tag{10}$$

This equation allows us to solve the original inference in the mean-field approximation in a single step. To determine the marginals of the empirical freqiencies, we just need to determine the empirical connected correlation matrix and invert it to get the couplings $e_{ij}$.

## C Adaptive MCMC Sampling in adabmDCA

To ensure accurate estimation of the marginals, adabmDCA monitors both equilibration and decorrelation of the Markov chains through sequence overlaps. For two sampled sequences $s_n^i, s_m^k \in \mathcal{A}^L$, the normalized overlap is defined as

$$O\big(s_n^i, s_m^k\big) = \frac{1}{L}\sum_{j=1}^{L} \delta\big(s_n^i(j), s_m^k(j)\big) \tag{11}$$

Three types of overlaps are used:

- External overlap ($O_{\text{ext}}$): between samples from different chains at the same time $n$.
- Internal-1 overlap ($O_{\text{int1}}$): between consecutive samples of the same chain at times $n$ and $n+1$.
- Internal-2 overlap ($O_{\text{int2}}$): between samples of the same chain at times $n$ and $n+2$.

At each epoch, adabmDCA computes averages $\mu_\alpha$ and standard errors $\sigma_\alpha$ ($\alpha \in \{\text{ext}, \text{int1}, \text{int2}\}$). In equilibrium, all three overlaps should agree within statistical error.

**Adaptive update of $T_{\text{wait}}$**

To control correlation between successive samples, $T_{\text{wait}}$ is adjusted dynamically:

Increase $T_{\text{wait}}$ (double it) if:

$$|\mu_{\text{ext}} - \mu_{\text{int2}}| > 5\sqrt{\sigma_{\text{ext}}^2 + \sigma_{\text{int2}}^2} \tag{12}$$

i.e. samples at lag $2T_{\text{wait}}$ are still too correlated.

Reduce $T_{\text{wait}}$ (average with last pre-increase value) if:

$$|\mu_{\text{ext}} - \mu_{\text{int1}}| < 5\sqrt{\sigma_{\text{ext}}^2 + \sigma_{\text{int1}}^2} \tag{13}$$

i.e. decorrelation already occurs at lag $T_{\text{wait}}$. Equilibration time is then set to $T_{\text{eq}} = 2T_{\text{wait}}$.

This adaptive scheme ensures that chain samples are equilibrated, initial bias is removed after $T_{\text{eq}}$ steps, and decorrelated, independence between samples at lag $T_{\text{wait}} - 2T_{\text{wait}}$. As a result, the estimated marginals match those of the equilibrium Potts distribution, guaranteeing stable gradient estimates and convergence of the Boltzmann machine.