universe

IMPACT FACTOR 2.6

CITESCORE 5.2

Article

# Deep Learning Spinfoam Vertex Amplitudes: The Euclidean Barrett–Crane Model

Hanno Sahlmann and Waleed Sherif

Special Issue

Loop Quantum Gravity and Non-Perturbative Approaches to Quantum Cosmology, Second Edition

Edited by
Prof. Dr. Gerald B. Cleaver

MDPI

https://doi.org/10.3390/universe11070235 *universe*

MDPI

*Article*

# Deep Learning Spinfoam Vertex Amplitudes: The Euclidean Barrett–Crane Model

**Hanno Sahlmann and Waleed Sherif ***

Institute for Quantum Gravity, Department of Physics, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Staudtstraße 7, 91058 Erlangen, Germany; hanno.sahlmann@fau.de

* Correspondence: waleed.sherif@fau.de

## Abstract

Spinfoam theories propose a well-defined path-integral formulation for quantum gravity, and it is hoped that they will provide the dynamics of loop quantum gravity. However, it is computationally hard to calculate spinfoam amplitudes. The well-studied Euclidean Barrett–Crane model provides an excellent setting for testing analytical and numerical tools to probe spinfoam models. We explore a data-driven approach to accelerating spinfoam computations by showing that the vertex amplitude is an object that can be learned from data using deep learning. We divide the learning process into a classification and a regression task: Two networks are independently engineered to decide whether the amplitude is zero or not and to predict the precise numerical value, respectively. The trained networks are tested with several accuracy measures. The classifier in particular demonstrates robust generalisation far outside the training domain, while the regressor demonstrates high predictive accuracy in the domain it is trained on. We discuss limitations, possible improvements, and implications for future work.

**Keywords:** loop quantum gravity; spinfoams; deep learning; numerical calculations

check for updates

Academic Editor: Gerald B. Cleaver

Received: 12 June 2025
Revised: 13 July 2025
Accepted: 18 July 2025
Published: 19 July 2025

**Citation:** Sahlmann, H.; Sherif, W. Deep Learning Spinfoam Vertex Amplitudes: The Euclidean Barrett–Crane Model. *Universe* **2025**, *11*, 235. https://doi.org/10.3390/universe11070235

## 1. Introduction

Loop quantum gravity (LQG) is a canonical quantisation program for general relativity (GR) that attempts to keep general covariance as manifest as possible [1–4]. In its kinematical sector, the theory is well-formulated: a Hilbert space $\mathscr{H}_{kin}$ spanned by spin-network states—labelled by graphs whose oriented edges carry irreducible SU(2) representations and whose vertices carry SU(2) invariant intertwiners [3]. The dynamics, however, are far more elusive. It is encoded in constraints—operators imposing equations (or, in the Master constraint approach [5–7], a single equation) on the physical states. Solutions are timeless a priori. Finding solutions and giving them a spacetime interpretation remains a formidable open problem despite various efforts (see, for example, [8–16]).

Path integral formulations of gravity have been considered for a long time since they avoid the spacetime split inherent in the canonical approach. Spinfoam (SF) theories (for example, see [17–26]), in particular, originated from the observation that gravity can be obtained by breaking the symmetries of a topological gauge theory. These theories side-step part of the difficulties of canonical LQG by constructing transition amplitudes between spin-network states as a sum over (discrete quantum) histories.

To regularise the gravitational path integral, one triangulates the manifold by a simplicial complex $T$ and works on its dual 2-complex $\Delta^*$. A spinfoam is precisely such a 2-complex whose faces $f$ are labelled by irreps $\pi_f$ of the gauge group and whose edges $e$

carry intertwiners $i_e$ between the irreps of the faces meeting in $e$. The regularised partition function can be written in factorised form [19,20]

$$Z_{\Delta^*} = \sum_{\pi_f, i_e} \prod_f A_f(\pi_f) \prod_e A_e(i_e) \prod_v A_v(\pi_f, i_e), \tag{1}$$

where $v$, $e$ and $f$ are dual to 4-simplices, tetrahedra and triangles of $T$, respectively. Mirroring the role of a vertex in ordinary Feynman diagrams, the vertex amplitude $A_v$ encodes the local dynamics of quantum geometry [27]. The precise nature of the gauge group, $\pi_f$, $i_e$ and $A_v$ depends on the spacetime dimension, signature and theory.

A serious problem is that $Z_{\Delta^*}$ is very hard to calculate in practice. The sum contains infinitely many terms in principle. Moreover, even calculating the vertex amplitude $A_v$ can be difficult, since it typically is a highly oscillatory function, which involves multidimensional sums or integrals in its definition and depends on the data of the vertex-spin network, i.e., the $\pi_f$- and $i_e$-label surrounding $v$.

The development of various numerical methods [28], each tailored to suit different regimes, have played a pivotal role in driving advancements in spinfoam models. Highly optimised, high-performance computing libraries such as `sl2cfoam` [29] and its successor `sl2cfoam-next` [30,31] enable efficient evaluation of Engle–Pereira–Rovelli–Livine (EPRL) amplitudes [27,32] and have been used to study various aspects of spinfoam models [33–35]. Monte Carlo methods, enhanced by Lefschetz thimble techniques to deform integration contours, have also been utilised in the context of spinfoams [36]. Different sampling methods of representation labels, such as importance sampling and random sampling of the bulk spins, as well as utilising generative flow networks, have been utilised to further enhance the convergence of spinfoam amplitudes involving several simplices or efficiently computing expectation values of observables [37,38]. Monte Carlo methods were also used to compute the vertex amplitude for a given set of coherent states as boundary data [39]. Recently, tensor-network methods and techniques from many-body quantum physics have been utilised to significantly reduce both the computational complexity and memory requirements for computing vertex amplitudes for both SU(2) and EPRL spinfoam vertex amplitudes [40].

On the large-spin or asymptotic regime, numerical programs such as the complex critical points program [41–43] have been developed to identify semi-classical geometries in the limit of large representations, while a hybrid program was proposed in [44] to bridge between the quantum and semi-classical regimes. Numerical methods [45,46], based on symmetry reductions, have also been employed in the study of the renormalisation aspects of spinfoam models.

Despite this considerable effort in both analytical and numerical approaches (see also [47–52]), the computation of vertex amplitudes remains a challenging problem. This motivates the exploration of alternative approaches to address the inherent computational complexity. The goal of the present work is to establish a new approach to accelerating spinfoam computations by showing that the vertex amplitude is an object that can be learned from the data using deep learning. In the simplest form, a trained neural network would approximate $A_v$ by interpolating from training data. In a more ambitious form, the neural network would learn to extrapolate much beyond its training domain in certain spin regimes. Further, another aim of this work is to complement the numerical implementations of exact analytical methods by helping identify dominant configurations, guiding importance sampling and enabling efficient pre-selection in possibly large parameter spaces.

While neural networks have been utilised in the context of spinfoam computations to calculate expectation values of observables [38] by learning the boundary configurations which contribute large amplitudes, the current work, as far as we are aware, is the first of its kind where the vertex amplitude itself is learned using a neural network. Consequently, we aim for a proof-of-principle, not an application to state-of-the-art spinfoam models. To keep things as non-technical as possible, we consider a very well-studied and non-trivial model, the Euclidean Barrett–Crane model (BC model) [53–57]. While interesting, this model is nowadays understood to be unphysical [58]. The action for the path integral for this theory derives from the Plebanski (or $BF$-theory plus simplicity constraints) action [59]

$$S_{Plebanski}[B, \omega, \lambda] = \int \mathrm{tr}[B \wedge F(\omega) + \lambda B \wedge B], \tag{2}$$

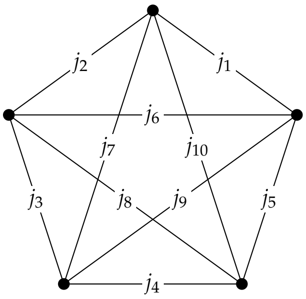
where $B$ is an so(4) -valued 2-form, $\omega$ is an so(4) connection, and $\lambda$ is a matrix-valued Lagrange multiplier enforcing the simplicity constraints that reduce topological $BF$-theory to the familiar Palatini action [55,57].[1]

Exploiting $\mathrm{Spin}(4) \cong \mathrm{SU}(2)_+ \times \mathrm{SU}(2)_-$, the BC model strongly imposes the simplicity constraints, thereby restricting every face to a balanced representation $(j_+, j_-) = (j, j)$. While this drastic reduction yields a manageable state sum, it also freezes the intertwiner degrees of freedom, eliminating physical degrees of freedom and effectively rendering the model unphysical [58]. More refined models, such as the EPRL or Engle–Pereira–Rovelli–Livine–Freidel–Krasnov (EPRL-FK) [60] models, impose the constraints weakly. They reintroduce an SU(2) intertwiner label, depend non-trivially on the Barbero–Immirzi parameter $\gamma$ and reproduce the BC model (intertwiner) in the limits $\gamma \to \pm\infty$ [25].

We will consider the BC model on a simplicial complex. Its vertex amplitude then takes the characteristic and compact form [61]

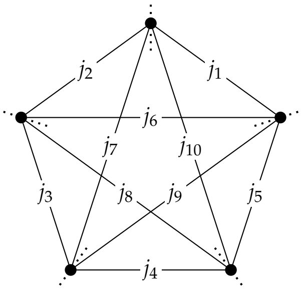
$$A_v(j_f) = \left(\prod_f (2j_f + 1)^k\right)\{10j\}, \tag{3}$$

where $\{10j\}$ denotes the Riemannian $10j$ symbol and $k$ an integer, which parametrises the choice of triangle weight in the measure.[2] The Riemannian $10j$ symbol is a function of 10 spins obtained by evaluating the following closed SU(2) × SU(2) spin-network on a flat connection [62]

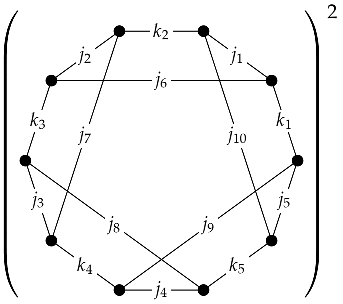


(4)

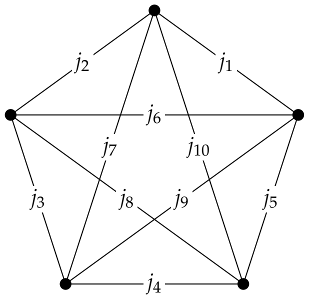
where the vertices are labelled by Barrett–Crane intertwiners. One can also define a modified $10j$ symbol using the modified Barrett–Crane intertwiners as the weighted sum of SU(2) spin-networks [62]. Since the value of a disjoint union of spin-networks is simply the product of their values, it follows that the modified $10j$ symbol is [62]



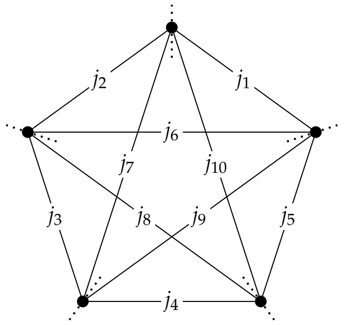
$$= \sum_{k_1,\dots,k_5} \left( \prod_{i=1}^{5} (2k_i + 1) \right) \tag{5}$$



Given the choice of splitting shown above, the modified $10j$ symbol can be shown to be always non-negative and is related to the $10j$ symbol by [62]



$$= (-1)^{2(j_1+\cdots+j_{10})} \tag{6}$$



Given that $\{10j\}$ is the vertex amplitude in this model, several algebraic and numerical algorithms [63–65] for evaluating $\{10j\}$, as well as its large spin asymptotics, have been extensively developed and studied, yet the computational cost still scales unfavourably with the magnitudes of the spins.

In this work, we will explore a data-driven alternative: casting the evaluation of the $10j$ symbol, and thus the vertex amplitude of the Barrett–Crane model, as a supervised learning problem for deep neural networks. Concretely, we (i) generate a comprehensive training set of exact $10j$ symbol values using an optimised algorithm, (ii) break down the learning task into classification and regression tasks and (iii) quantify the fidelity and generalisation of the learned amplitude on spins that lie outside the training domain. The goals of this work are twofold. First, we demonstrate a *proof-of-principle*: (high dimensional) vertex amplitudes of spinfoam models are amenable to modern deep learning techniques. Second, we lay the groundwork for accelerating numerical investigations of more realistic models. Because the EPRL/FK vertex reduces to its BC counterpart in specific limits, the aim is for the models and tools used and developed in this work to be ported, with appropriate modifications capturing the $\gamma$-dependence, to the state-of-the-art SF models.

The presentation of the work is as follows:

(i) In Section 2, we start by clearly stating the goals of the current work, as well as presenting the methodology to be used.
(ii) In Section 2.1, we discuss the generation of training data and the pre-processing of the produced datasets prior to the training process.
(iii) Section 2.2 discusses the specific architecture of the networks used in this work and encoding schemes used to facilitate the learning process.
(iv) Section 2.3 concerns the training protocols for both the classification and regression tasks conducted in this work, as well as the evaluation metrics to which we evaluate the networks post-training.
(v) In Section 3, we present the results of the training for both classification and regression tasks.

(vi) Lastly, in Section 4, we review the issues encountered in this work, the limitations of the current work and how it relates to future models to be considered and present some avenues for exploration of these problems.

## 2. Methodology

As mentioned, this study serves as a proof-of-principle to address the question of whether or not a neural network can be utilised to learn the vertex amplitude of a given spinfoam model, in this case, the Euclidean Barrett–Crane model. The approach we use is one of supervised learning (SL). We approach the problem in terms of two tasks, which can be roughly outlined as follows. First, a classification task where, given a data set $\{\mathbf{S}, A(\mathbf{S})\}$ of spin configurations $\mathbf{S}$ and corresponding amplitudes $A(\mathbf{S})$, we train a classifer $\mathscr{C}(\mathbf{S})$ to determine whether the corresponding amplitude $A(\mathbf{S})$ is zero or not. Next, a regression task where, once again, given the dataset described above, we train a regressor $\mathscr{R}(\mathbf{S})$ to predict the correct amplitude $A(\mathbf{S})$ for the given spin configuration $\mathbf{S}$. We then construct a meta "Expert" network $\mathscr{P}(\mathbf{S})$, which combines both $\mathscr{C}(\mathbf{S})$ and $\mathscr{R}(\mathbf{S})$ to provide the correct predicted amplitude for the given configuration $\mathbf{S}$.

We will proceed by detailing the components of the implementation in natural order. That is, we first briefly describe data acquisition and pre-processing, followed by the network architectures and encoding schemes. Next, we outline the training protocol, hyper-parameter choices and monitored metrics. Following that, for each task, we present the results for the mentioned evaluation metrics. A discussion regarding limitations, technical details and ablation studies is conducted in the Discussion (see Section 4).

### 2.1. Data Generation and Processing

The core component of the vertex amplitude in the Euclidean Barrett–Crane model is the $\{10j\}$, which can be negative, positive or zero. To simplify our task, we will focus on learning the square of the $\{10j\}$ for both classification and regression. For classification, the appropriate sign factor can be easily reconstructed from the given spin configuration. This effectively allows us to reduce the complexity of the classification task while remaining able to reconstruct the correct sign of the learned $\{10j\}$. For regression, the square root of the prediction can be taken at inference. The tools and software used in this work are all Python based. As such, to facilitate the training process, the algorithm presented in [63] and the corresponding implementation in C provided therein[3] has been rewritten in Python and accelerated by utilising just-in-time compilation using `numba` [66] to obtain a comparatively fast compile time compared to the C implementation.

For both tasks, we respectively train the networks within a specified spin cutoff. That is, for a given cutoff spin $j_{max}$, then the spin configurations to be considered are ones such that $(j_1, j_2, \cdots, j_{10}) =: \mathbf{S}^{\mathbf{j_{max}}} \in (\mathscr{S}^{(j_{max})})^{10}$, where $\mathscr{S}^{(j_{max})} = \left\{ j \in \frac{1}{2}\mathbb{Z} | j \leq j_{max} \right\}$, $|\mathscr{S}^{(j_{max})}| = K := 2j_{max} + 1$ and the superscript is explicitly stated in $\mathbf{S}^{(\mathbf{j_{max}})}$ to indicate the cutoff to which the configuration belongs. The following datasets are created:

(i) For classification: a dataset $\mathscr{D}_{\mathscr{C}}^{(j_{max})} := \left\{ \mathbf{S}^{(\mathbf{j_{max}})}, \sigma(\mathbf{S}^{(\mathbf{j_{max}})}) \right\}$ containing pairs of spin configurations $\mathbf{S}^{(\mathbf{j_{max}})}$ and their corresponding $\{10j\}^2$ signs denoted as $\sigma(\mathbf{S}^{(\mathbf{j_{max}})}) := \mathrm{sgn}(\{10j\}(\mathbf{S}^{(\mathbf{j_{max}})}))^2$. Since $(\{10j\}(\mathbf{S}^{(\mathbf{j_{max}})}))^2 \in \mathbb{R}_{\geq 0}$, then for any $\mathbf{S}^{(\mathbf{j_{max}})}$, it follows that $\sigma(\mathbf{S}^{(\mathbf{j_{max}})}) \in [0, 1]$. This dataset contains data points for *all* possible configurations $\mathbf{S}^{(\mathbf{j_{max}})}$ in a given cutoff.

(ii) For regression: another dataset $\mathscr{D}_{\mathscr{R}}^{(j_{max})} := \left\{ \mathbf{S}^{(\mathbf{j_{max}})}, \mathrm{log10j}(\mathbf{S}^{(\mathbf{j_{max}})}) \right\}$, which then contains pairs of spin configurations $\mathbf{S}^{(\mathbf{j_{max}})}$ and their corresponding $\mathrm{log10j}(\mathbf{S}^{(\mathbf{j_{max}})}) := \log\Big( (\{10j\}(\mathbf{S}^{(\mathbf{j_{max}})}))^2 + \epsilon \Big)$, where $\epsilon = 1 \times 10^{-26}$ is a small correction factor. Unlike

$\mathscr{D}_{\mathscr{C}}^{(j_{max})}$, this dataset contains data points for all configurations $\mathbf{S}^{(\mathbf{j_{max}})}$ in a given cutoff, *which have a non-zero* $\log 10\mathrm{j}(\mathbf{S}^{(\mathbf{j_{max}})})$ *value*. Thus, $|\mathscr{D}_{\mathscr{R}}^{(j_{max})}| < |\mathscr{D}_{\mathscr{C}}^{(j_{max})}|$ always.

For small enough cutoffs, such datasets can be obtained simply by full enumeration of the space $\mathscr{S}^{(j_{max})}$. To understand the distribution of the potential training data, one can compare the number of non-zero $\log 10\mathrm{j}(\mathbf{S}^{(\mathbf{j_{max}})})$ for all $\mathbf{S}^{(\mathbf{j_{max}})}$ in different cutoffs.

It is immediately evident, then, as shown in Figure 1, that there is an imbalance in the number of configurations, which yield a zero amplitude compared to non-zero amplitudes. In fact, as the cutoff increases, the percentage of non-zero amplitude configurations increases drastically, reaching over 60% already at $j_{max} = 2.5$. Further, the value of $(\{10j\}(\mathbf{S}^{(\mathbf{j_{max}})}))^2$ can vary by more than 22 orders of magnitude in even small cutoffs as $j_{max} = 2.0$. This, therefore, poses the following hurdles: (i) a class imbalance for classification and (ii) a very large range in the values of $\log 10\mathrm{j}(\mathbf{S}^{(\mathbf{j_{max}})})$ for the regression.

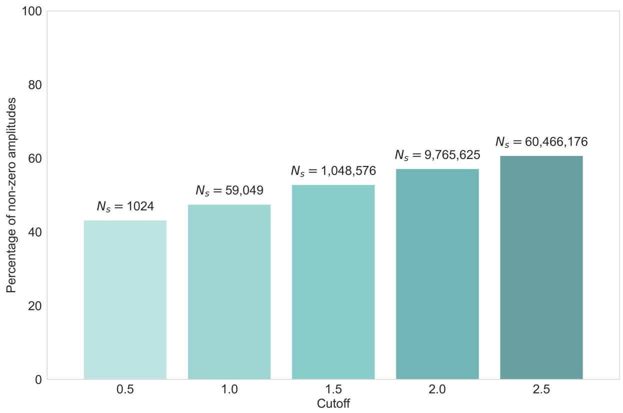


**Figure 1.** Percentage of non-zero to zero vertex amplitudes at different cutoffs, with the total number of states $N_s$ shown for each cutoff.

### 2.2. Network Architecture and Encoding Schemes

As this study is a proof-of-principle, little emphasis was put on constructing a sophisticated network architecture for either task. Remarkably, as will be shown, simple architectures suffice for both tasks. A multi-layer perceptron (MLP) [67,68] was used for both the classification and regression tasks. For classification, the classifier $\mathscr{C}(\mathbf{S})$ consisted of an MLP with one hidden layer (depth 1) and 128 hidden nodes (width 128). A rectified linear unit (ReLU) activation function

$$\mathrm{ReLU}(x) = \max(0, x) \tag{7}$$

was used to introduce the non-linearity. The input for the classifier $\mathscr{C}(\mathbf{S})$ was simply the spin configuration $\mathbf{S}^{(\mathbf{j_{max}})}$. The architecture for the classifier remained constant irrespective of the training cutoff. The classifier maintained a number of trainable parameters $\texttt{Params}(\mathscr{C})$ of only 1537 parameters.

For the regressor $\mathscr{R}(\mathbf{S})$, an MLP was used as well. Unlike $\mathscr{C}(\mathbf{S})$, the depth and width of $\mathscr{R}(\mathbf{S})$ varied depending on the $j_{max}$ chosen during training. For example, at $j_{max} = 1.0$, $\mathscr{R}(\mathbf{S})$ had a depth of 6 and a width of 256, while for $j_{max} = 1.5$, it had a depth of 5 and a width of 512. Different activation functions were considered and tested. Ultimately, a Gaussian error linear units activation function was used

$$\mathrm{GELU}(x) = x\Phi(x), \tag{8}$$

where $\Phi(x)$ is the cumulative distribution function for the Gaussian distribution. Unlike ReLU activation, the derivative of the GELU function is continuous at the origin. A smooth activation allows the optimiser to track higher-order curvature and avoids the kinks that piece-wise linear units, such as ReLU, introduce. Further, while ReLU either passes the signal unchanged ($x$) or blocks it (0), the GELU multiplies the input $x$ by the probability that a unit drawn from a standard Gaussian is below $x$. Small inputs are only tempered, not annihilated, which preserves information carried by low-magnitude features (inputs).

While the classifier $\mathscr{C}(\mathbf{S})$ takes as input the configuration $\mathbf{S}$, the regressor $\mathscr{R}(\mathbf{S})$ does not. The input for $\mathscr{R}(\mathbf{S})$ is encoded using an indexing function $\Omega$. For a single $j \in \mathcal{S}^{(j_{max})}$, let $\mathbf{i} : \mathcal{S}^{(j_{max})} \to \{0, 1, \ldots, K-1\}$ such that $\mathbf{i}(j) = 2j$. Let $\vec{e}_r \in \mathbb{R}^K$ be the standard $r$-th basis vector such that $(\vec{e}_r)_q = \delta_{q,r}$. We define the one-hot encoding of a single spin to be the map $\omega : \mathcal{S}^{(j_{max})} \to \{0,1\}^K \subset \mathbb{R}^K$ such that $\omega(j) = \vec{e}_{\mathbf{i}(j)}$, which places a 1 exactly at the coordinate that corresponds to the value of $j$. The one-hot encoding for a configuration $\mathbf{S}^{\mathbf{j_{max}}}$ is then $\Omega : (\mathcal{S}^{(j_{max})})^{10} \to \{0,1\}^{10K} \subset \mathbb{R}^{10K}$. Simply put, while the classifier takes as input a configuration of 10 values in a given $j_{max}$, the regressor takes as input $10K$ values at the same $j_{max}$.

The motive behind the encoding is driven by empirical observations during training. Namely, the network performance was poor for un-encoded inputs due to it attempting to associate the inputs in a purely arithmetic manner (e.g., the network prediction being merely the mean of the inputs). Consequently, we convert every spin to a pure indicator vector rather than feed its numeric value directly. While this may not always be an optimal encoding depending on the problem, it is, nevertheless, sufficient for our case, as it removes any potential artificial ordinality and avoids adding arbitrary distances, relations or embeddings that the network might exploit as if they were physically meaningful, allowing it to view the spin values merely as categorical symbols. The encoding nevertheless provides compatibility with transfer learning across cutoffs. Thus, the chosen encoding provides a minimal, symmetry-respecting[4] representation that supports generalisation and compatibility across cutoffs. Note that one can, in principle, choose any faithful encoding map, not necessarily restricted to the one mentioned above, and this may play a crucial role in the performance of the network. Other encodings, such as learned embeddings or sinusoidal schemes, could be explored in future work depending on the target model and computational constraints.

### 2.3. Training Protocol and Hyperparameter Choices

This work utilises PyTorch [69] for all neural network-related aspects. A non-exhaustive preliminary assessment conducted via automatic hyperparameter tuning using `optuna` [70] was carried out to determine an estimate for the best hyperparameters in this work. The following parameters, shared for both tasks, were thus obtained. The networks in both tasks were trained with a mini-batch AdamW optimiser [71] with a weight decay of $1 \times 10^{-6}$ and a 1-cycle learning rate schedule with a peak step size of $1 \times 10^{-3}$ for the case of the regressor and $1 \times 10^{-4}$ for the case of the classifier. The batch size for the classification task was 8, while a batch size of 256 was chosen for the regression task. For both tasks, training takes place in the low spin regime, as the behaviour of the network evaluations can be extrapolated to further training on higher spins. In what follows, we outline task-specific evaluation metrics and further training protocols.

### 2.3.1. Classification Metrics and Protocol

In what follows, we often refer to the $\sigma(\mathbf{S}^{(\mathbf{j_{max}})})$ for a given $\mathbf{S}^{(\mathbf{j_{max}})}$ as a *label*. In this binary classification task, the label can either be 0 or 1. The binary classification task was optimised with respect to a weighted binary cross-entropy loss function

$$\text{BCELoss} = -\frac{1}{N}\sum_{i=1}^{N}[\omega_1 y_i \log p_i + \omega_0(1-y_i)\log(1-p_i)], \tag{9}$$

as standard for such classification tasks, where $y_i$ denotes the actual binary label (0 or 1) of the $i$-th observation, $p_i$ denotes the probability of the $i$-th observation to be in class 1 and $N$ is the total number of observations made. Note that $\omega_0$ and $\omega_1$ are class weights used to neutralise the imbalance between zero and non-zero amplitudes. This is done by amplifying the contribution of underrepresented classes during training. Such a weighted scheme is one way to ensure that the network maintains strong performance across both precision and recall, especially for higher cutoffs where the proportion of non-zero configurations increases. As will be shown in the sections that follow, the evaluation metrics confirm that this approach successfully mitigates the imbalance. Note that $\omega_0$ and $\omega_1$ can simply be chosen to be inversely proportional to the respective class frequencies.

The training protocol is done cutoff-wise. First, a network is trained and evaluated at a cutoff of 0.5. Since the network architecture is static between cutoffs, transfer learning was utilised and the network was then retrained, starting with previously optimised parameters from the preceding cutoff, on a cutoff of 1.0 and then once again evaluated. This protocol continued until a cutoff of 2.0. The training dataset size starts at 75% of all available configurations at a cutoff of 0.5. As transfer learning was conducted successively for higher cutoffs, the training dataset size was decreased incrementally to establish a minimal dataset size: the smallest dataset size required to achieve the highest evaluation metrics values. Note that the datasets were collected blindly, and no active binning in terms of cutoffs was employed. While this is possible, it has not been done intentionally to test the model's capacity to train from blind data.

Several metrics were evaluated after each cutoff training cycle as follows:

(i) Hard accuracy: this is defined as

$$\text{AccHard} = \frac{1}{N}\sum_{i=1}^{N} 1\cdot\{\mathtt{f}(\hat{y}_i) = y_i\}, \tag{10}$$

which essentially counts, on the test sample of size $N$, how many network predicted labels $\hat{y}_i$ exactly match the true labels $y_i$. Here, $\mathtt{f}(\hat{y}_i)$ is a decision threshold function which returns 1 if $\hat{y}_i \geq 0.5$ and 0 otherwise.

(ii) Soft accuracy: this is defined as

$$\text{AccSoft} = \frac{1}{N}\sum_{i=1}^{N}(1 - |\hat{y}_i - y_i|) \tag{11}$$

which essentially computes how "far off" the network prediction was from the true labels for a given test batch of size $N$.

(iii) Precision [72]: which is defined as

$$P = \frac{TP}{TP+FP}, \tag{12}$$

where $TP$ denotes true positives (data points with labels 1, which have been correctly predicted by the network), and $FP$ denotes false positives (data points with labels 0, which have been incorrectly predicted by the network to have label 1), which conveys the fraction of all predicted non-zero states that are correct. Essentially, this metric focuses on the quality of positive predictions, giving a measure of how trustworthy the network's prediction is when predicting non-zero labels.

(iv) Recall [72]: the ability to correctly predict all non-zero configurations, which is defined as

$$R = \frac{TP}{TP + FN}, \tag{13}$$

where $FN$ denotes false negatives (data points with labels 1, which have been incorrectly predicted by the network to have a label 0). Having a high recall value indicates that the network rarely fails to correctly label a non-zero configuration.

(v) $F_1$ score [72]: defined as

$$\mathrm{F}_1 = 2\frac{P \cdot R}{P + R}, \tag{14}$$

is the harmonic mean of precision and recall, symmetrically representing both precision and recall in one metric. The highest obtainable F-1 score of 1.0 indicates perfect precision and recall.

Additionally, for every cutoff training round, a confusion matrix, evaluated on the entire configuration space, is computed. This gives a clear picture of the number of FN and FP values for the network.

### 2.3.2. Regression Metrics and Protocol

Unlike the classification task, the regressor was trained on a single chosen cutoff. No transfer learning was utilised. Therefore, a regressor $\mathscr{R}$ trained at a cutoff of 0.5 would only be evaluated on states $\mathbf{S}^{(\mathbf{0.5})}$. This is due to the dynamic nature of the network's architecture used in this task, which makes transfer learning, although not impossible, much harder. For all cutoffs, the training data consisted of 85% of all available non-zero configurations $\mathbf{S}^{(\mathbf{j_{max}})}$ at the current cutoff. Since the labels in this task can have a rather large range, the training dataset was not blindly collected. Rather, the entire space was first enumerated, after which the amplitudes were binned according to their magnitude. The produced dataset of 85% of all non-zero contributing data points was stratified according to those bins, ensuring sufficient representation across all magnitudes available.

The loss function for this task was chosen to be the Huber loss [73]

$$\mathrm{HuberLoss}(e) = \begin{cases} \frac{1}{2}e^2 & , \quad |e| < \delta \\ \delta(|e| - \frac{1}{2}\delta) & , \quad |e| \geq \delta \end{cases} \tag{15}$$

where $e := \widehat{\log10j}(\mathbf{S}^{(\mathbf{j_{max}})}) - \log10j(\mathbf{S}^{(\mathbf{j_{max}})})$. For the case of $\delta = 1$, this is equivalent to the smooth L1 loss function. Here, $\widehat{\log10j(\mathbf{S}^{(\mathbf{j_{max}})}})$ denotes the network predicted log value of the square of the $\{10j\}$ symbol of the given configuration. For brevity, we will denote that with $\hat{y}_{\log}$ and denote the true value with $y_{\log}$. For any training cutoff, the following metrics were observed after training:

(i) Root mean squared error (RMSE) in log space: this is simply defined as $\mathrm{RMSE}_{\log} = \sqrt{\mathrm{MSE}_{\log}}$ where

$$\mathrm{MSE}_{\log} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_{\log} - y_{\log})^2. \tag{16}$$

Note that this metric may be sensitive to errors in large-valued labels. Applying it in log space is due to the large possible label magnitudes, which span several orders. This then ensures that large values do not disproportionately dominate the error.

(ii) Median absolute deviation (MAD) in log space: defined as

$$\mathrm{MAD}_{\log} = \mathrm{Median}\left( \left\{ |\hat{y}_{\log}^{(i)} - y_{\log}^{(i)}| \right\}_{i=1}^{N} \right) \tag{17}$$

is a measure of error also in the log space. Unlike the $\mathrm{RMSE}_{\log}$, this metric is resilient to outliers and better reflects the typical prediction deviation.

(iii) Mean absolute percentage error (MAPE): is given as

$$\mathrm{MAPE} = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{e^{\hat{y}_{\log}} - e^{y_{\log}}}{e^{y_{\log}}} \right|, \tag{18}$$

which can be interpreted in terms of the relative error of predictions over a test set of size $N$. Note that this metric is not computed in log space.

(iv) Threshold accuracy: lastly, we measure the threshold accuracy, which is

$$\mathrm{Acc}_{\leq \epsilon} = \frac{1}{N} \sum_{i=1}^{N} 1 \cdot \left\{ |e^{\hat{y}_{\log}} - e^{y_{\log}}| \leq \epsilon |e^{y_{\log}}| \right\} \tag{19}$$

which is an indicator of how many predictions in a test set of size $N$ have a relative error lower than a specified $\epsilon$ threshold. In this work, we take $\epsilon = 0.1$, and thus, the $\mathrm{Acc}_{\epsilon}$ will measure how many predictions fall within a 10% relative error to the true value. Once again, this metric is not computed in log space.

Additionally, we also compute the $R^2$ value, as well as a true vs. prediction plot, after every cutoff training cycle.

## 3. Training Results

The two criteria being sought in this work for both tasks are (i) whether the networks can perform well at the cutoff they are trained in and (ii) whether the networks can predict on test samples from a cutoff they have not been trained on. All computations were carried out on an Intel Xeon E3-1240 v5 with 4 cores at 3.5 GHz, and no distributed, parallelised or GPU computations were utilised. We begin with the classification task. Carrying out the protocol described in Section 2.3.1, the results shown in Table 1 were observed.

**Table 1.** The training and test loss for the classification task on different cutoffs is shown. Here, $N_s$ denotes the total number of possible configurations at the cutoff and $N_{\text{train}}$ denotes the number of samples used in the training process.

| $j_{max}$ | Training Loss | Test Loss | Training Time (s) | $N_s$ | $N_{\text{train}}$ | $N_{\text{train}}/N_s$ |
|---|---|---|---|---|---|---|
| 0.5 | 0.01772 | 0.15405 | 59.7 | 1024 | 768 | 0.75 |
| 1.0 | $8.5601 \times 10^{-7}$ | 0.0362 | 2807.28 | 59,049 | 36,126 | 0.611 |
| 1.5 | $1.49516 \times 10^{-7}$ | 0.01092 | 13,386.76 | 1,048,576 | 103,618 | 0.098 |
| 2.0 | 0.00236 | 0.02397 | 24,134.8 | 9,765,625 | 170,356 | 0.017 |

Table 1 shows the training and test loss for the classification task on different cutoffs. As shown, the training dataset size starts at 75% of the total number of available configurations at the cutoff $N_s$ for $j_{max} = 0.5$. As transfer learning is applied and training proceeds to higher cutoffs, the training dataset size decreases until 1.17% for $j_{max} = 2.0$. Despite that, both training and test loss decrease steadily as the cutoff increases, indicating that the learning process is carried out successfully. The increase in the test and training loss for $j_{max} = 2$ is attributed to the relatively small training dataset size.

After each training cycle, the trained network was tested on all configurations for cutoffs $j_{max} \in [0.5, 1.0, 1.5, 2.0, 2.5]$. The following metrics in Table 2 were observed.

**Table 2.** Classification metrics for the classifier trained with transfer learning at different cutoffs.

| Training $j_{max}$ | Test $j_{max}$ | AccSoft (%) | AccHard (%) | $P$ | $R$ | $F_1$ |
|---|---|---|---|---|---|---|
| 0.5 | 0.5 | 96.7432 | 98.8281 | 0.9837 | 0.9773 | 0.9804 |
| | 1.0 | 83.3652 | 83.9506 | 0.7197 | 0.8216 | 0.7672 |
| | 1.5 | 81.0237 | 81.3578 | 0.6973 | 0.8139 | 0.7511 |
| | 2.0 | 79.7331 | 79.9418 | 0.6951 | 0.7989 | 0.7433 |
| | 2.5 | 78.8568 | 78.9951 | 0.6974 | 0.7844 | 0.7383 |
| 1.0 | 0.5 | 99.9996 | 100 | 1.0 | 1.0 | 1.0 |
| | 1.0 | 99.9392 | 99.9395 | 0.9989 | 0.9989 | 0.9989 |
| | 1.5 | 98.9591 | 98.9833 | 0.9780 | 0.9929 | 0.9853 |
| | 2.0 | 96.3990 | 96.4228 | 0.9223 | 0.9845 | 0.9523 |
| | 2.5 | 94.2651 | 94.2786 | 0.8823 | 0.8792 | 0.8792 |
| 1.5 | 0.5 | 99.9999 | 100 | 1.0 | 1.0 | 1.0 |
| | 1.0 | 99.9719 | 99.9712 | 0.9996 | 0.9995 | 0.9995 |
| | 1.5 | 99.9369 | 99.9382 | 0.9994 | 0.9988 | 0.9990 |
| | 2.0 | 99.5092 | 99.5187 | 0.9908 | 0.9960 | 0.9933 |
| | 2.5 | 98.2248 | 98.2348 | 0.9638 | 0.9905 | 0.9769 |
| 2.0 | 0.5 | 99.9994 | 100 | 1.0 | 1.0 | 1.0 |
| | 1.0 | 99.9773 | 99.9762 | 0.9995 | 0.9997 | 0.9995 |
| | 1.5 | 99.9666 | 99.9681 | 0.9996 | 0.9994 | 0.9994 |
| | 2.0 | 99.8993 | 99.9028 | 0.9986 | 0.9988 | 0.9986 |
| | 2.5 | 99.5982 | 99.6070 | 0.9966 | 0.9930 | 0.9947 |

As shown in Table 2, all classification metrics are within an excellent range for the cutoff that the network has been trained on. However, one striking aspect is that the simple classifier seems to be able to perform reasonably well when tested on higher cutoffs, which it has not been trained on. Additionally, no catastrophic forgetting[5] was observed. Further, as the classifier is trained on a higher cutoff, using transfer learning, the metrics for test cutoffs that fall above the training cutoff also improve drastically. For example, a classifier trained at $j_{max} = 0.5$ and shows an F-1 score of 0.7383 when tested on $j_{max} = 2.5$, while the same classifier retrained using transfer learning up to $j_{max} = 2.0$ has an F-1 score of 0.9947 for a test cutoff of $j_{max} = 2.5$. This can be further elucidated by looking at the confusion matrices for the classifier at different training stages, as shown in Figure 2.

Figure 2 shows the confusion matrices for the classifier at different stages of training, tested on different cutoffs at each stage. In a given confusion matrix, the top left quadrant denotes the true negatives, the top right quadrant denotes the false negatives, the bottom left quadrant denotes the false positives and the bottom right quadrant denotes the true positives. It is evident that as the classifier progresses in the training stages, the number of false predictions in either class becomes lower. This can be more easily shown by looking at only the false negatives and false positives, as shown in Figure 3.

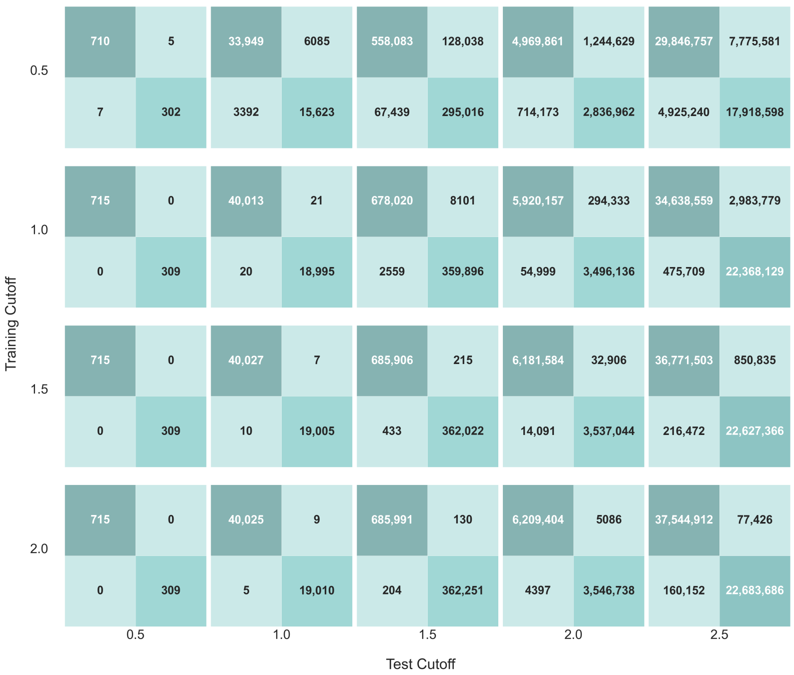
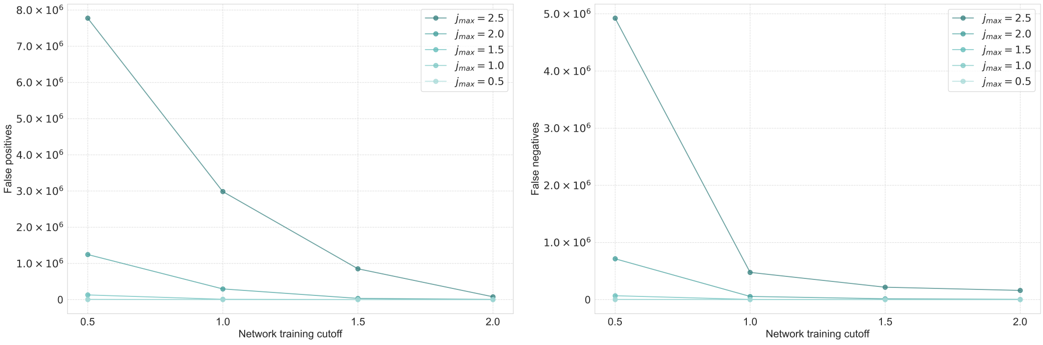


**Figure 2.** The confusion matrices for the classifier at different stages of training, tested on different cutoffs each time after each training round.

As can be seen in Figure 3, both the false positives and false negatives fall drastically. Perhaps most interestingly, the number of false negatives and false positives is seen to decrease even on cutoffs on which the classifier has not been trained on (see the trend-line of $j_{max} = 2.5$ in both figures), as the classifier training progresses to higher cutoffs. The takeaway here is that despite the simple architecture of the classifier and the extremely small size of training datasets for higher cutoffs, the classifier is able to generalise well to cutoffs beyond its training.



(**a**) False Positives (**b**) False Negatives

**Figure 3.** The number of false positives and false negatives for the classifier trained at different cutoffs and tested on cutoffs from 0.5 to 2.5 is shown in (**a**) and (**b**), respectively.

As discussed in the protocol in Section 2.3.2, we approach the regression task differently from the classification. In what follows, we mainly focus on a regressor trained at cutoffs of 1.0 and 1.5. In the discussion section (Section 4), we elaborate on higher cutoffs.

A regressor with a depth of 6 and width of 256 with a GELU activation was used for training at $j_{max} = 1.0$, while for $j_{max} = 1.5$, the regressor had a depth and width of 5 and 512, respectively. In both cases, training was conducted over 200 epochs, and the training dataset consisted of 85% of all configurations, which yield a non-zero $\{10j\}^2$ (16,162 and

308,086 data points for the cutoffs 1.0 and 1.5, respectively). No transfer learning was applied. The figure below shows the training loss for both regressors.

As shown in Figure 4, no irrecoverable or sustained spikes were observed. The regression metrics were observed to be as shown in Table 3.

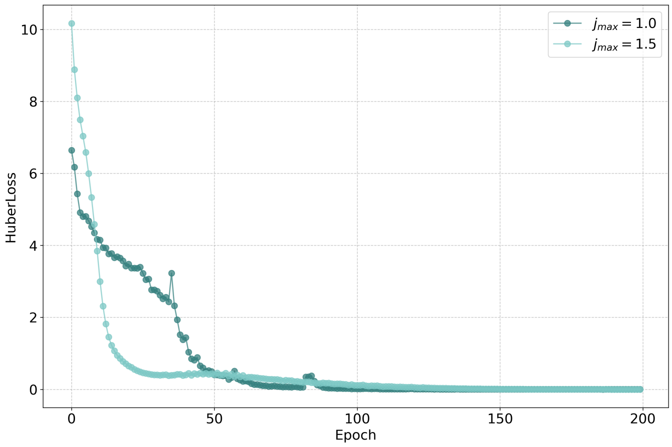


**Figure 4.** The training loss during the regressor training at cutoffs of 1.0 and 1.5.

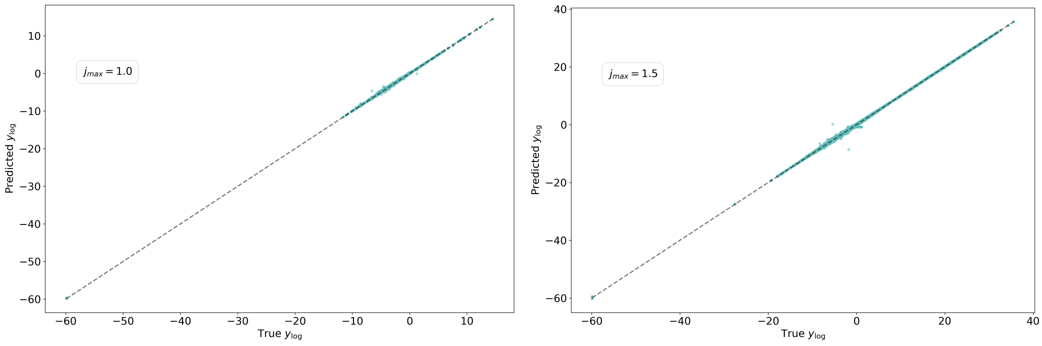
As shown in Table 3, all metrics fall within a good range for such regression tasks. The regressor trained at a cutoff of 1.0 demonstrates excellent predictive accuracy, achieving a MAPE of 2.7587% and a threshold accuracy $\text{Acc}_{\leq 0.1}$ of 94.3045%, indicating that the vast majority of predictions fall within a 10% relative error margin of the true values when evaluating the model on all possible configurations, which yield non-zero labels at the cutoff. Further, its $\text{RMSE}_{\text{log}}$ and $\text{MAD}_{\text{log}}$ values reflect a low dispersion of residuals and highlight the consistency of the regressor's output. The high $R^2$ score implies that nearly all variance in the target variable is captured by the regressor.

**Table 3.** Evaluation metrics on all non-zero configurations for the regressors trained at different cutoffs.

| Training Cutoff | MAPE (%) | $\text{Acc}_{\leq 0.1}$ (%) | $\text{RMSE}_{\text{log}}$ | $\text{MAD}_{\text{log}}$ | $R^2$ |
|---|---|---|---|---|---|
| 1.0 | 2.7587 | 94.3045 | 0.0582 | 0.0215 | 0.9986 |
| 1.5 | 4.1735 | 83.6901 | 0.0851 | 0.0396 | 0.9999 |

In comparison, the model trained at $j_{max} = 1.5$ exhibited slightly less accurate performance, but nevertheless, yielded promising results. The MAPE and $\text{Acc}_{\leq 0.1}$ values indicate that it maintains a good predictive fidelity. While the $\text{RMSE}_{\text{log}}$ and $\text{MAD}_{\text{log}}$ values are comparatively higher, they are still within acceptable bounds. Notably, the $R^2$ value for this regressor is even higher, indicating a better fit for the data. To further elucidate upon the performance of both regressors, a True vs. Prediction plot can be shown below.

Figure 5 shows the True vs. Prediction plots in log space for both regressors at the cutoff of 1.0 and 1.5 on the left and right, respectively. The data points evaluated in the plots constitute all data points at the cutoff which yield a non-zero label. As shown, most of the predicted labels align well with the true labels in both cases, with only a few predictions which fall far from the true label values. Overall, the regressors seem to be performing relatively well. No minimal training dataset investigation or transfer learning was applied in either case.



(**a**) Cutoff 1.0 (**b**) Cutoff 1.5

**Figure 5.** True vs. Prediction plot in log space for the regressors trained at cutoffs (**a**) 1.0 and (**b**) 1.5. The data points shown include all non-zero-labelled data at the respective cutoffs.. Note that the dashed line indicates the region where the prediciton matches the true value exactly.

We also note that the trained regressors did not generalise well beyond their training domain. This can be attributed to different reasons. For example, a regressor initially trained on a dataset of some cutoff $j_1$ and subsequently fine-tuned on a dataset from the following cutoff $j_2$ is effectively exposed to only a small fraction of the $j_1$ data during transfer learning. This is due to the fact that the dataset corresponding to $j_2$ subsumes the $j_1$ dataset, but also includes an overwhelmingly larger volume of new samples, causing the regressor to be prone to catastrophic forgetting, wherein knowledge acquired during the initial training phase may be overwritten or degraded during fine-tuning.

This effect may also be caused, or at least exacerbated, by the use of one-hot encoding, which treats each input category as orthogonal and independent. As new categories may be introduced in the $j_2$ dataset, the model may assign high importance to these new features, diminishing the influence of earlier, sparsely repeated categories from the $j_1$ dataset. In all cases, such a behaviour is to be expected, as extrapolation is generally a non-trivial task and is made more difficult by the nature of the vertex amplitude functions being highly oscillatory in general.

*Expert Network*

So far, the task of computing the vertex amplitude using neural networks has been divided into classification and regression tasks. Further, we have focused on learning the sign of the $(\{10j\}(\mathbf{S}^{(\mathrm{j_{max}})}))^2$ for the classification task and the value of $\mathrm{log10j}(\mathbf{S}^{(\mathrm{j_{max}})})$ for the regression task. To produce the correct amplitude value for a given spin configuration, we need to:

(i) At inference time, exponentiate the regressor's output to obtain $(\{10j\}(\mathbf{S}^{(\mathrm{j_{max}})}))^2$ instead of $\mathrm{log10j}(\mathbf{S}^{(\mathrm{j_{max}})})$ and then further take the square root to obtain the correct $\{10j\}(\mathbf{S}^{(\mathrm{j_{max}})})$ value.

(ii) Insert the correct sign factor based on the spins in the given $\mathbf{S}^{(\mathrm{j_{max}})}$, as shown in Equation (6) to obtain the correct sign of the computed $\{10j\}$.

(iii) Insert the correct positive dimensionality multiplicative factor related to the spins in the given $\mathbf{S}^{(\mathrm{j_{max}})}$, as shown in Equation (3) to compute the complete vertex amplitude of the BC model.

As such, the last piece of the work includes creating a meta network, denoted $\mathscr{P}(\mathbf{S})$, which combines both the classifier and the regressor. The output of this meta network, which we shall call an Expert, is merely the product of the outputs of $\mathscr{C}(\mathbf{S})$ and $\mathscr{R}(\mathbf{S})$, along with all the relevant corrections mentioned above.

The reason for this is not only aesthetic. The grouping of such networks into an Expert $\mathscr{P}$ allows for having an ensemble of Experts for a given cutoff, each trained with different seeds and therefore different datasets, in the hope of increasing the predictive accuracy of the overall model and producing more precise error estimation. Further, one can combine Experts in a Mixture of Experts (MoE) approach [74,75], which, in simple terms, houses within it $M$ experts (or $M$ ensembles of $N$ experts each), each trained at a different cutoff. After appropriate training of the gating in such an MoE, this then results in one model that can accept any configuration that falls within the range of cutoffs it has been trained on and yield an accurate prediction based on the Experts it contains. This, however, will be left for future work.

## 4. Discussion

One of the most, and perhaps the most, pressing inherent issues in this work is the scarcity of data. Generally, SL is a greedy approach, and this only gets worse if the objective we attempt to learn is complicated (e.g., very large range, highly non-uniform, very sensitive to the inputs). This was already observed during the training of the regressor in this simple toy model. Different training methods, including transfer learning, different one-hot encoding and dataset processing, resulted in either catastrophic forgetting or low predictive accuracy. This, however, can of course be due to the network architecture. Nevertheless, the issue of the regression task being data greedy is expected to still persist. In that case, one needs to tailor the networks such that they require as few data points as possible. Further, data collection can be conducted by Reinforcement Learning (RL) methods. During this work, we have also collected data points by training an agent with a proximal policy optimisation algorithm (PPO) [76] to find the highest valued amplitudes without exhaustive enumeration of all possible configurations. This can then cut down on unnecessarily computing vertex amplitudes, which may be irrelevant to the desired training process, while still being a computationally expensive process.

Spinfoam vertex amplitudes, for a given set of coherent states as boundary data, have been computed using Monte Carlo methods [39]. One may be able to adapt the generative flow networks approach [38] (initially used to compute expectation values of observables by learning the regions in which the amplitude is large) to facilitate a similar computation, as done in [39]. Such flow networks may also provide another “agent”-like approach for data collection in the domain of direct learning of the vertex amplitude itself, as we have carried out. How to exactly set it up is not clear at the current time, but an interesting avenue to explore in future work.

Ultimately, we recognise that this work serves only as a proof-of-principle, and thus, we refrained from exploring or utilising all possible avenues to identify, resolve or optimise such issues and bottlenecks. This is left to be conducted for later work on more physically relevant models, but we are aware that this is a persistent issue in the nature of this approach. The resolution of this issue will also largely rely, in part, on utilising other efficient numerical methods to generate a sufficient amount of data for training, as this sets the bound of the amount of data available for training.

The next pressing issue is the regression problem. The learned data for the regressor in this work spanned a very large range, which only grew with the cutoff. This is due to the nature of the learned amplitude, as they are generally represented by highly oscillatory functions. This poses several serious concerns. For example, assuming the data acquisition process is not an issue (may already be not true for more realistic models such as the EPRL model at high spin), devising a training set which has enough representatives from each order of magnitude is a non-trivial task. This will also highly depend on the network and loss function used, as different architectures and losses can be less sensitive to very large data, while others might require more large data representatives in the dataset. It is therefore easy to see how this becomes a concern if one can not enumerate the entire space.

In the cutoffs presented in this work, the classifier maintained a static architecture, which included a number of trainable parameters $\texttt{Params}(\mathscr{C})$ of only 1537 parameters across all training. Despite that, it was demonstrated that it excelled in learning whether the given spin configuration would yield a non-zero squared $\{10j\}$ or not. The regressors, on the other hand, had a number of trainable parameters $\texttt{Params}(\mathscr{R})$ of 350,093 and 1,091,725 for $j_{max} = 1.0$ and 1.5, respectively. Compared to the total number of non-zero data points available at the same cutoffs (19,015 and 362,455, respectively), one sees that this approach is simply inefficient: with enough learnable parameters, one can fit any data. In this case, it is much faster to simply create a table of all possible amplitude values for all spins in the current cutoff. If a regressor trained on some cutoff $j_i$ can, to some degree of acceptable evaluation metrics, predict on a higher cutoff $j_k > j_i$, then $\texttt{Params}(\mathscr{R})$ being larger than the number of the available states at the training cutoff can be overlooked. This, however, is not the case in this work.

Nevertheless, that does not mean that it is not possible. Ablation studies (here discussed for $m_{max} = 1.0$) were conducted where different MLPs with different widths and depths were tested. It was observed that one can get to moderately acceptable evaluation metrics with MLPs of depth 3 and width 64 ($\texttt{Params}(\mathscr{R}) = 10,685$) and even MLPs with depth 1 and width 256 ($\texttt{Params}(\mathscr{R}) = 8253$). While in both cases $\texttt{Params}(\mathscr{R})$ is less than the total number of available data points, this does not immediately translate to all relevant evaluation metrics being consistently high or the training process being conducted smoothly. Further, different architectures, such as recurrent neural networks with gating[6] were observed to be good candidates for the task. Lastly, given the graph-based nature of the $10j$ symbol, graph neural networks (GNNs) [78] might also serve as a good candidate. However, this work has not explored such an architecture. Lastly, the regressors in this work did not produce satisfactory evaluation metrics when evaluated beyond the training domain. This is unsurprising, as this is an inherent limitation to such tasks, which is further made difficult by the objective function being learned, here the vertex amplitude, being of a difficult nature.

This leads to the following conclusion: while the regressors used in this work are inefficient, *in principle*, we believe that there are more efficient architectures to be explored with $\texttt{Params}(\mathscr{R})$ being less than the available training data. What they are, how well they train and whether they, if at all possible, generalise for higher cutoffs or not were not tasks of principal importance in this work. The reason being that it is unclear precisely how the tools developed in this work would translate to physically relevant models such as the EPRL model. The purpose of this work is to merely demonstrate a proof-of-principle, and exhaustive studies will be left for later work.

## 5. Conclusions

Spinfoam theories provide dynamics for non-perturbative loop quantum gravity by constructing transition amplitudes between spin-network states through a sum over their histories. The quantisation procedure implements simplicity constraints at the quantum level, resulting in a regularised partition function $Z_{\Delta^*}$ on a spinfoam $\Delta^*$. One of the main components of $Z_{\Delta^*}$ is the vertex amplitude $A_v$, which, akin to QED, encodes the local dynamics of quantum geometry. In this work, we investigated the feasibility, as a proof-of-principle, of a data-driven approach, whereby the vertex amplitude $A_v^{BC}$ of the Euclidean Barrett–Crane model is learned using deep neural networks, specifically the Riemannian $10j$ symbol, which is at the core of $A_v^{BC}$ in this model. The amplitude is learned through a two-step process whereby first, a classifier $\mathscr{C}$ is trained to predict whether, for a given set of spin configurations $\mathbf{S}$, the resulting $A_v^{BC}(\mathbf{S})$ is zero or not. Second, a regressor $\mathscr{R}$ is trained to predict the exact numerical value of the $\{10j\}^2$. As a last step, we construct a meta network, which we denote as an Expert $\mathscr{P}$, which utilises both $\mathscr{C}$ and $\mathscr{R}$ to construct the correct amplitude $A_v^{BC}$ by inserting the relevant sign and dimensionality factors.

For the classification task, a small MLP was trained on several cutoffs ranging from 0.5 to 2.0, each time utilising transfer learning. Despite the relative training dataset size being reduced from 75% to roughly 1% of all available states for cutoffs of 0.5 to 2.0, respectively, the classifier $\mathscr{C}$ proved successful by being able to have high evaluation metrics (soft accuracy, hard accuracy, precision, recall and F-1 score) both within and well above the training cutoffs. The regression task proceeded with MLPs of dynamic architecture, which depended on the training cutoff. Within the learned cutoff, the regressors showed high evaluation metrics (RMSE, MAD in log space, MAPE and threshold accuracy with $\epsilon = 0.1$) in both cutoffs presented (1.0 and 1.5). Generalisation to cutoffs beyond the training regime for the regressor case, however, was unsuccessful. This may, preliminarily, be attributed to (i) catastrophic forgetting during transfer learning from one cutoff to the next, due to the overwhelming volume of new data in subsequent cutoffs relative to the prior ones and (ii) the use of one-hot encoding, which introduces sparsity and treats new categories (i.e., higher spins) as orthogonal, potentially amplifying the model's focus on newly introduced features at the expense of earlier ones. Therefore, the choice of encoding scheme may significantly influence any potential generalisation performance of such regressors.

An Expert $\mathscr{P}$ was constructed to correctly output the full amplitude $A_v^{BC}$. We also discuss the limitations and hurdles encountered during this work, mainly the generalisation of the regressor to higher cutoffs and the training process with limited data. We propose an ensemble approach to increase the accuracy for the trained cutoffs and a reinforcement learning-inspired approach to collect relevant training data, where an agent is trained using a proximal policy optimisation algorithm to learn which configurations $\mathbf{S}$ would yield amplitudes relevant to the training process at hand. We concluded this work by elucidating different network architectures which may be better suited for the regression task. However, as this work stands as a proof-of-principle, we refrain from exploring all avenues to resolve the issues encountered in this work; this will be pursued later for physically relevant models. Nevertheless, the current work stands as an addition to existing numerical methods to compute the vertex amplitude of spinfoam models and provides a proof-of-principle that vertex amplitudes of spinfoam models are amenable to modern deep learning techniques. The aim of the surrogate models to be developed in this data-driven approach is to complement the numerical implementations of exact analytical methods by helping identify dominant configurations, guiding importance sampling and Monte Carlo approximations and enabling efficient pre-selection in possibly large parameter spaces.

**Author Contributions:** Conceptualization, H.S. and W.S.; Methodology, H.S. and W.S.; Software, H.S. and W.S.; Validation, H.S. and W.S.; Formal analysis, H.S. and W.S.; Investigation, H.S. and W.S.; Resources, H.S. and W.S.; Data curation, H.S. and W.S.; Writing — original draft, H.S. and W.S.; Writing — review and editing, H.S. and W.S.; Visualization, H.S. and W.S.; Supervision, H.S. and W.S.; Project administration, H.S. and W.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The data presented in this study are openly available in Zenodo at https://doi.org/10.5281/zenodo.15649600.

**Acknowledgments:** HS thanks Wojciech Kaminski and Martin Zeiß for many helpful conversations about spinfoam models. HS acknowledges the contribution of the COST Action CA23130. The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). The hardware is funded by the German Research Foundation (DFG). This research was supported in part by Perimeter Institute for Theoretical Physics. Research at Perimeter Institute is supported by the Government of Canada through the Department of Innovation, Science and Economic Development and by the Province of Ontario through the Ministry of Research, Innovation and Science.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Notes

[1] For the Lorentzian signature, one replaces SO(4) with SO(1, 3) and must deal with non-compact representations. We confine ourselves to the Euclidean sector throughout this work.

[2] The SF formalism suggests using the measure that gives exactly a topologically invariant partition function before imposing the constraints reducing $BF$-theory to GR, which amounts to choosing $k = 2$.

[3] `http://jdc.math.uwo.ca/spin-foams/10j-code/`, (accessed 2 April 2025).

[4] The chosen encoding should not, in principle, make it more difficult for the network to recognise symmetries of the underlying learned objective, in this case, the symmetries of the spin-network the $\{10j\}$ is defined on.

[5] The classifier forgetting about previously learned cutoffs after being trained on a higher cutoff.

[6] Networks with gated recurrent units (GRUcells) [77].

## References

1. Rovelli, C. Loop quantum gravity. *Living Rev. Rel.* **1998**, *1*, 1. [CrossRef] [PubMed]
2. Thiemann, T. Modern canonical quantum general relativity. *arXiv* **2001**. [CrossRef]
3. Ashtekar, A.; Lewandowski, J. Background independent quantum gravity: A Status report. *Class. Quant. Grav.* **2004**, *21*, R53. [CrossRef]
4. Han, M.; Huang, W.; Ma, Y. Fundamental structure of loop quantum gravity. *Int. J. Mod. Phys. D* **2007**, *16*, 1397–1474. [CrossRef]
5. Thiemann, T. The Phoenix project: Master constraint program for loop quantum gravity. *Class. Quant. Grav.* **2006**, *23*, 2211–2248. [CrossRef]
6. Dittrich, B.; Thiemann, T. Testing the master constraint programme for loop quantum gravity. I. General framework. *Class. Quant. Grav.* **2006**, *23*, 1025–1066. [CrossRef]
7. Thiemann, T. Quantum spin dynamics. VIII. The Master constraint. *Class. Quant. Grav.* **2006**, *23*, 2249–2266. [CrossRef]
8. Thiemann, T. Quantum spin dynamics (QSD). *Class. Quant. Grav.* **1998**, *15*, 839–873. [CrossRef]
9. Thiemann, T. Quantum spin dynamics (qsd). 2. *Class. Quant. Grav.* **1998**, *15*, 875–905. [CrossRef]
10. Varadarajan, M. Anomaly free quantum dynamics for Euclidean LQG. *arXiv* **2022**. [CrossRef]
11. Bojowald, M. Absence of singularity in loop quantum cosmology. *Phys. Rev. Lett.* **2001**, *86*, 5227–5230. [CrossRef] [PubMed]
12. Ashtekar, A.; Pawlowski, T.; Singh, P. Quantum Nature of the Big Bang: Improved dynamics. *Phys. Rev. D* **2006**, *74*, 084003. [CrossRef]
13. Guedes, T.L.M.; Mena Marugán, G.A.; Müller, M.; Vidotto, F. Taming Thiemann's Hamiltonian constraint in canonical loop quantum gravity: Reversibility, eigenstates and graph-change analysis. *arXiv* **2024**, arXiv:2412.20272.
14. Guedes, T.L.M.; Mena Marugán, G.A.; Vidotto, F.; Müller, M. Computing the graph-changing dynamics of loop quantum gravity. *arXiv* **2024**, arXiv:2412.20257. [CrossRef]
15. Sahlmann, H.; Sherif, W. Towards quantum gravity with neural networks: Solving the quantum Hamilton constraint of U(1) BF theory. *Class. Quant. Grav.* **2024**, *41*, 225014. [CrossRef]
16. Sahlmann, H.; Sherif, W. Towards quantum gravity with neural networks: Solving quantum Hamilton constraints of 3d Euclidean gravity in the weak coupling limit. *Class. Quant. Grav.* **2024**, *41*, 215006. [CrossRef]
17. Reisenberger, M.P. World sheet formulations of gauge theories and gravity. *arXiv* **1994**. [CrossRef]
18. Reisenberger, M.P.; Rovelli, C. 'Sum over surfaces' form of loop quantum gravity. *Phys. Rev. D* **1997**, *56*, 3490–3508. [CrossRef]
19. Rovelli, C.; Vidotto, F. *Covariant Loop Quantum Gravity: An Elementary Introduction to Quantum Gravity and Spinfoam Theory*; Cambridge University Press: Cambridge, UK, 2014; ISBN 978-1-107-06962-6/978-1-316-14729-0.
20. Perez, A. The Spin Foam Approach to Quantum Gravity. *Living Rev. Rel.* **2013**, *16*, 3. [CrossRef] [PubMed]
21. Baez, J.C. Spin foam models. *Class. Quant. Grav.* **1998**, *15*, 1827–1858. [CrossRef]
22. Bambi, C.; Modesto, L.; Shapiro, I. *Handbook of Quantum Gravity*; Springer: Berlin/Heidelberg, Germany, 2024; ISBN 978-981-99-7680-5/978-981-99-7681-2/978-981-19-3079-9. [CrossRef]
23. Engle, J.; Speziale, S. *Spin Foams: Foundations*; Springer: Berlin/Heidelberg, Germany, 2023. [CrossRef]

24. Livine, E.R. Spinfoam Models for Quantum Gravity: Overview. In *Encyclopedia of Mathematical Physics*; Elsevier: Amsterdam, The Netherlands, 2025. [CrossRef]
25. Kaminski, W.; Kisielowski, M.; Lewandowski, J. Spin-Foams for All Loop Quantum Gravity. *Class. Quant. Grav.* **2010**, *27*, 095006; Erratum in: *Class. Quant. Grav.* **2012**, *29*, 049502. [CrossRef]
26. Bahr, B.; Hellmann, F.; Kaminski, W.; Kisielowski, M.; Lewandowski, J. Operator Spin Foam Models. *Class. Quant. Grav.* **2011**, *28*, 105003. [CrossRef]
27. Engle, J.; Pereira, R.; Rovelli, C. The Loop-quantum-gravity vertex-amplitude. *Phys. Rev. Lett.* **2007**, *99*, 161301. [CrossRef] [PubMed]
28. Dona, P.; Han, M.; Liu, H. *Spinfoams and High-Performance Computing*; Springer: Berlin/Heidelberg, Germany, 2023. [CrossRef]
29. Donà, P.; Fanizza, M.; Sarno, G.; Speziale, S. Numerical study of the Lorentzian Engle-Pereira-Rovelli-Livine spin foam amplitude. *Phys. Rev. D* **2019**, *100*, 106003. [CrossRef]
30. Gozzini, F. A high-performance code for EPRL spin foam amplitudes. *Class. Quant. Grav.* **2021**, *38*, 225010. [CrossRef]
31. Dona, P.; Frisoni, P. How-to Compute EPRL Spin Foam Amplitudes. *Universe* **2022**, *8*, 208. [CrossRef]
32. Engle, J.; Livine, E.; Pereira, R.; Rovelli, C. LQG vertex with finite Immirzi parameter. *Nucl. Phys. B* **2008**, *799*, 136–149. [CrossRef]
33. Donà, P.; Gozzini, F.; Sarno, G. Numerical analysis of spin foam dynamics and the flatness problem. *Phys. Rev. D* **2020**, *102*, 106003. [CrossRef]
34. Donà, P.; Frisoni, P.; Wilson-Ewing, E. Radiative corrections to the Lorentzian Engle-Pereira-Rovelli-Livine spin foam propagator. *Phys. Rev. D* **2022**, *106*, 066022. [CrossRef]
35. Frisoni, P.; Gozzini, F.; Vidotto, F. Markov chain Monte Carlo methods for graph refinement in spinfoam cosmology. *Class. Quant. Grav.* **2023**, *40*, 105001. [CrossRef]
36. Han, M.; Huang, Z.; Liu, H.; Qu, D.; Wan, Y. Spinfoam on a Lefschetz thimble: Markov chain Monte Carlo computation of a Lorentzian spinfoam propagator. *Phys. Rev. D* **2021**, *103*, 084026. [CrossRef]
37. Donà, P.; Frisoni, P. Summing bulk quantum numbers with Monte Carlo in spin foam theories. *Phys. Rev. D* **2023**, *107*, 106008. [CrossRef]
38. Bunao, J.; Frisoni, P.; Kogios, A.; Wogan, J. Generative Flow Networks in Covariant Loop Quantum Gravity. *arXiv* **2024**, arXiv:2407.19036. [CrossRef]
39. Steinhaus, S. Monte Carlo algorithm for spin foam intertwiners. *Phys. Rev. D* **2024**, *110*, 026022. [CrossRef]
40. Asante, S.K.; Steinhaus, S. Efficient tensor network algorithms for spin foam models. *Phys. Rev. D* **2024**, *110*, 106018. [CrossRef]
41. Han, M.; Huang, Z.; Liu, H.; Qu, D. Complex critical points and curved geometries in four-dimensional Lorentzian spinfoam quantum gravity. *Phys. Rev. D* **2022**, *106*, 044005. [CrossRef]
42. Han, M.; Liu, H.; Qu, D. Complex critical points in Lorentzian spinfoam quantum gravity: Four-simplex amplitude and effective dynamics on a double-Δ3 complex. *Phys. Rev. D* **2023**, *108*, 026010. [CrossRef]
43. Han, M.; Liu, H.; Qu, D. Mathematica program for numerically computing real and complex critical points in four-dimensional Lorentzian spinfoam amplitudes. *Phys. Rev. D* **2025**, *111*, 024021. [CrossRef]
44. Asante, S.K.; Simão, J.D.; Steinhaus, S. Spin-foams as semiclassical vertices: Gluing constraints and a hybrid algorithm. *Phys. Rev. D* **2023**, *107*, 046002. [CrossRef]
45. Bahr, B.; Steinhaus, S. Investigation of the spinfoam path integral with quantum cuboid intertwiners. *Phys. Rev. D* **2016**, *93*, 104029. [CrossRef]
46. Bahr, B.; Steinhaus, S. Numerical evidence for a phase transition in 4d spin foam quantum gravity. *Phys. Rev. Lett.* **2016**, *117*, 141302. [CrossRef] [PubMed]
47. Asante, S.K.; Dittrich, B.; Haggard, H.M. Effective Spin Foam Models for Four-Dimensional Quantum Gravity. *Phys. Rev. Lett.* **2020**, *125*, 231301. [CrossRef] [PubMed]
48. Asante, S.K.; Dittrich, B.; Padua-Arguelles, J. Effective spin foam models for Lorentzian quantum gravity. *Class. Quant. Grav.* **2021**, *38*, 195002. [CrossRef]
49. Asante, S.K.; Dittrich, B.; Haggard, H.M. Discrete gravity dynamics from effective spin foams. *Class. Quant. Grav.* **2021**, *38*, 145023. [CrossRef]
50. Dittrich, B. Modified graviton dynamics from spin foams: The area Regge action. *Eur. Phys. J. Plus* **2024**, *139*, 651. [CrossRef]
51. Dittrich, B.; Kogios, A. From spin foams to area metric dynamics to gravitons. *Class. Quant. Grav.* **2023**, *40*, 095011. [CrossRef]
52. Borissova, J.N.; Dittrich, B. Towards effective actions for the continuum limit of spin foams. *Class. Quant. Grav.* **2023**, *40*, 105006. [CrossRef]
53. Barrett, J.W.; Crane, L. Relativistic spin networks and quantum gravity. *J. Math. Phys.* **1998**, *39*, 3296–3302. [CrossRef]
54. Barrett, J.W.; Crane, L. A Lorentzian signature model for quantum general relativity. *Class. Quant. Grav.* **2000**, *17*, 3101–3118. [CrossRef]
55. Baez, J.C. An Introduction to Spin Foam Models of *BF* Theory and Quantum Gravity. *Lect. Notes Phys.* **2000**, *543*, 25–93. [CrossRef]
56. Reisenberger, M.P. On relativistic spin network vertices. *J. Math. Phys.* **1999**, *40*, 2046–2054. [CrossRef]

57. Livine, R.E.; Oriti, D. Barrett-Crane spin foam model from generalized BF type action for gravity. *Phys. Rev. D* **2002**, *65*, 044025. [CrossRef]
58. Alesci, E.; Rovelli, C. The Complete LQG propagator. I. Difficulties with the Barrett-Crane vertex. *Phys. Rev. D* **2007**, *76*, 104012. [CrossRef]
59. Plebanski, J.F. On the separation of Einsteinian substructures. *J. Math. Phys.* **1977**, *18*, 2511–2520. [CrossRef]
60. Freidel, L.; Krasnov, K. A New Spin Foam Model for 4d Gravity. *Class. Quant. Grav.* **2008**, *25*, 125018. [CrossRef]
61. Christensen, J.D.; Khavkine, I.; Livine, E.R.; Speziale, S. Sub-leading asymptotic behaviour of area correlations in the Barrett-Crane model. *Class. Quant. Grav.* **2010**, *27*, 035012. [CrossRef]
62. Baez, J.C.; Christensen, J.D. Positivity of spin foam amplitudes. *Class. Quant. Grav.* **2002**, *19*, 2291–2306. [CrossRef]
63. Christensen, J.D.; Egan, G. An Efficient algorithm for the Riemannian 10j symbols. *Class. Quant. Grav.* **2002**, *19*, 1185–1194. [CrossRef]
64. Baez, J.C.; Christensen, J.D.; Egan, G. Asymptotics of 10j symbols. *Class. Quant. Grav.* **2002**, *19*, 6489. [CrossRef]
65. Baez, J.C.; Christensen, J.D.; Halford, T.R.; Tsang, D.C. Spin foam models of Riemannian quantum gravity. *Class. Quant. Grav.* **2002**, *19*, 4627–4648. [CrossRef]
66. Lattner, C.; Adve, V. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO), San Jose, CA, USA, 21–24 March 2004; pp. 75–88.
67. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
68. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
69. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv* **2019**, arXiv:1912.01703.
70. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. *arXiv* **2019**, arXiv:1907.10902. [CrossRef]
71. Loshchilov, I.; Hutter, F. Decoupled Weight Decay Regularization. *arXiv* **2017**, arXiv:1711.05101.
72. Powers, D.M.W. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv* **2020**, arXiv:2010.16061. [CrossRef]
73. Huber, P.J. Robust Estimation of a Location Parameter. *Ann. Math. Stat.* **1964**, *35*, 73–101. [CrossRef]
74. Jacobs, R.A.; Jordan, M.I.; Nowlan, S.J.; Hinton, G.E. Adaptive Mixtures of Local Experts. *Neural Comput.* **1991**, *3*, 79–87. [CrossRef] [PubMed]
75. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; Dean, J. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017.
76. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
77. Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder—Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1724–1734.
78. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef] [PubMed]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.