xLSTM Networks for Top-Quark Jet Tagging

Jonathan Cavallin

***Abstract*—In particle physics, accurate jet tagging plays a crucial role, enabling the identification of events originating from different physical processes; a wide range of Deep Neural Network (DNN) approaches have been explored for this task. The current state of the art is represented by ParT, a transformer-based model. Recently, the xLSTM architecture has been introduced, a redesigned recurrent neural network that achieves performance comparable to, in some tasks even surpassing, transformer models in natural language modeling. In this work we present, to the best of our knowledge, the first application of xLSTM to jet tagging, focusing on the isolation of top-quark events. We describe the architecture and evaluate its performance on established jet tagging benchmarks, comparing it against simpler LSTM-based baselines. For this binary classification, the xLSTM JetTagger does not outperform standard LSTM architectures. Further work is needed to assess whether the architecture's potential can be fully exploited on the more complex problem of multi-class jet tagging, which is of particular interest for low-level trigger systems.**

***Index Terms*—xLSTM, Jet Tagging, LSTM, LHC**

I. Introduction

At the Large Hadron Collider (LHC) at CERN, beams of protons traveling close to the speed of light are brought to collide every 25ns, producing on average 600 million collisions per second. This results in the creation of a plethora of new unstable particles, which rapidly decay to form sprays of outgoing particles. In this context, a *jet* is a collimated spray of tens or hundreds of particles originating from a common collision (event). One of the most important analyses to perform is *jet tagging*, i.e., inferring which type of particle initiated a given reconstructed jet from the set of observables provided by the detectors. This information is crucial to isolate events originating from different physics processes, thereby enabling more precise measurements of the quantities of interest and enhancing the sensitivity to signs of new physics.

In this work, we adopt a sequence-based representation for jets, where each jet is described as an ordered sequence of its constituent particles. We progressively develop and evaluate the performance of recurrent neural network (RNN) architectures of increasing complexity, on the binary classification task of distinguishing top-quark events. We begin with a standard single-layer LSTM model, where the sequence output is aggregated through a simple mean pooling. Next, we extend this baseline by incorporating an attention mechanism, allowing the network to assign different weights to each particle in the jet. We then consider deeper architectures, specifically multi-layer LSTMs with attention, in order to capture more complex sequential dependencies. Finally, we investigate the recently proposed xLSTM architecture. We evaluate both a single-block xLSTM model and more sophisticated variants composed of different residual block sequences, and compare their performance against the aforementioned LSTM-based baselines.

Department of Physics and Astronomy, University of Padova, Email: jonathan.cavallin@studenti.unipd.it.

CloudVeneto is acknowledged for providing the computational resources used in this work.

II. Related Work

The introduction of Deep Neural Networks (DNNs) has played a fundamental role in advancing performances in the jet-tagging task, which can be naturally formulated as a multi-class classification problem. Within Deep Learning approaches, essentially three types of jet representations have been explored, each motivating different architectural choices for the proposed models:

- **Image-based representation:** jets are encoded as 2D images based on the energy deposited by each particle in the calorimeter cells used for detection. This approach, among the first presented, is naturally suited for different 2D CNN architectures, and was able to achieve significant improvements over previous non-ML, QCD-based multivariate methods [1]–[4]. The main drawback of this representation is the difficulty of incorporating additional, higher-level particle information beyond the calorimeter response.
- **Sequence-based representation:** jets are simply viewed as a collection of particles, each associated with a certain number of reconstructed features. Particles in each jet (sample) need to be sorted according to a certain metric, to be fed in order as a sequence of inputs of recurrent neural network (RNN) models [5]–[7]. Compared to image-based methods, it has the advantage of being more flexible, allowing to include any type of features describing the jet particles, and also offering a more compact representation. However, the choice of ordering may impact performance.
- **Particle cloud representation:** A special case of the previous, jets are represented by an unordered, variable-sized and permutation invariant sets of particles. Particle cloud representations have emerged as the dominant paradigm in recent years, driving the development of specialized architectures that explicitly exploit permutation symmetry. Building on this representation, ParticleNet [8] and the more recent transformer-based ParT [9] have established new state-of-the-art benchmarks in jet tagging performance.

Despite their excellent performance, transformer models come with a large number of trainable parameters (∼2.14M for ParT [9]), which makes their direct deployment in resource-constrained environments, such as level-1 triggers to be implemented on FPGAs, particularly challenging. As a result, these models are extremely powerful for offline analysis on collected and pre-processed datasets, but less suitable for fast triggering, where efficiency and latency are critical. This motivates the exploration of alternative architectures. In this context, it is of interest to perform a first test of xLSTMs which have shown good results in other domains and may provide a favorable balance between expressivity and computational efficiency.

III. Signals and Features

Dataset Overview. This work is based on the HLS4ML LHC Jet dataset, limited to jets with up to 30 particles, publicly available at this link. The dataset consists of simulated jets from proton-proton collisions at a center-of-mass energy of $\sqrt{s} \approx 13$ TeV. Data is obtained using the configuration and parametric description of LHC detectors to simulate the detection process. Jets are defined starting from individual constituent particles using the well-established anti-k_T algorithm with radius parameter $R = 0.8$.

Input Features and Preprocessing. For our study, we make use of the sequence-based representation (image-based is also available), in which each particle is described by 15 reconstructed kinematic features, summarized in Tab. 1. Each jet is therefore represented as a sequence (list) of up to 30 particles, each encoded as a 15-dimensional feature vector. For jets containing fewer than 30 particles, the sequence is zero-padded to a fixed length of 30. Consequently, each input sample can be considered as a tensor $\mathbf{x} \in \mathbb{R}^{30\times15}$.

To effectively exploit RNN architectures, it is necessary to define an ordering for the particle sequence provided to the network. Following previous works, in particular those performed on the same type of dataset [10], we order particles in descending order of transverse momentum p_T. Input samples are then standardized to zero mean and unit variance independently per feature.

Class Composition and Splitting. The dataset includes jets originating from light quarks (up, down, strange), gluons, W and Z bosons, and top quarks, corresponding to five distinct classes. In this work, we focus on the binary classification task of isolating top-quark jets (signal, labeled as $+1$) from all other jets, collectively treated as background (labeled as 0). The dataset is divided into a training set of 610k samples and a test set of 270k samples. We further randomly split the training set into the actual training set and a validation set, in a 90%-10% proportion. Hyperparameter tuning and early stopping, unless stated differently, are performed based on this validation set. In Fig.1 we present the distributions of the 15 features for the training set, separated into signal and background. Each value corresponds to the average over the constituent particles of each jet. Classes in the training set are imbalanced, with approximately a 4:1 ratio of background to signal; this imbalance is accounted for in the analysis.

Evaluation Metrics. Since the plain accuracy of the label prediction depends on the threshold chosen to discriminate between classes, it is not a suitable metric to use in the model optimization procedure. In a practical implementation of a tagger for an experiment, we would never use 0.5 as the classifier threshold. Instead, we typically define several working points corresponding to fixed thresholds for the false positive rate (FPR) tolerated; in this work we choose to evaluate performance at FPR = 10%, 1%, and 0.1% by reporting the corresponding true positive rate (TPR). A more significant metric for optimization is therefore the maximization of the area under the ROC curve (AUC in the following) of the model, which represents the probability that a randomly chosen positive sample will output a higher value than a randomly chosen background sample. Another commonly used metric [8], [9] we adopt is given by the inverse FPR (background rejection) at a given fixed TPR, which we denote Rej_{TPR}. In this work we use $\text{Rej}_{50\%}$ and $\text{Rej}_{30\%}$, which are common choices at LHC.

TABLE 1: Particle-level input features for each jet

Feature	Description
p_x, p_y, p_z	Momentum components
E	Energy
p_T	Transverse momentum
$\cos\theta$	Cosine of polar angle θ
η, ϕ	Pseudorapidity $\eta = -\ln(\tan(\theta/2))$ and azimuthal angle
ΔR	Distance from jet axis in the (η, ϕ) plane: $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$
E_{rel}, $p_{T,\text{rel}}$	Ratio of particle's energy and transverse momentum to its jet values
$\eta_{\text{rel}}, \phi_{\text{rel}}$	η, ϕ relative to the jet axis: $\eta_{\text{rel}} = \eta - \eta_{jet}$, $\phi_{\text{rel}} = \phi - \phi_{jet}$
$\eta_{\text{rot}}, \phi_{\text{rot}}$	Rotated η, ϕ as described in [10]

IV. Processing Pipeline

A. *xLSTM architecture*

We briefly summarize the architecture and main enhancements introduced by the novel xLSTM network [11]. Two different residual block types are presented, namely sLSTM and mLSTM, which modify standard LSTM by enhancing its memory structure and introducing exponential gating. The blocks differ in how memory storage and retrieval are organized. The detailed forward pass of the two blocks is presented in IV-B. By stacking such blocks in a residual fashion we obtain the basic architecture of a xLSTM model.

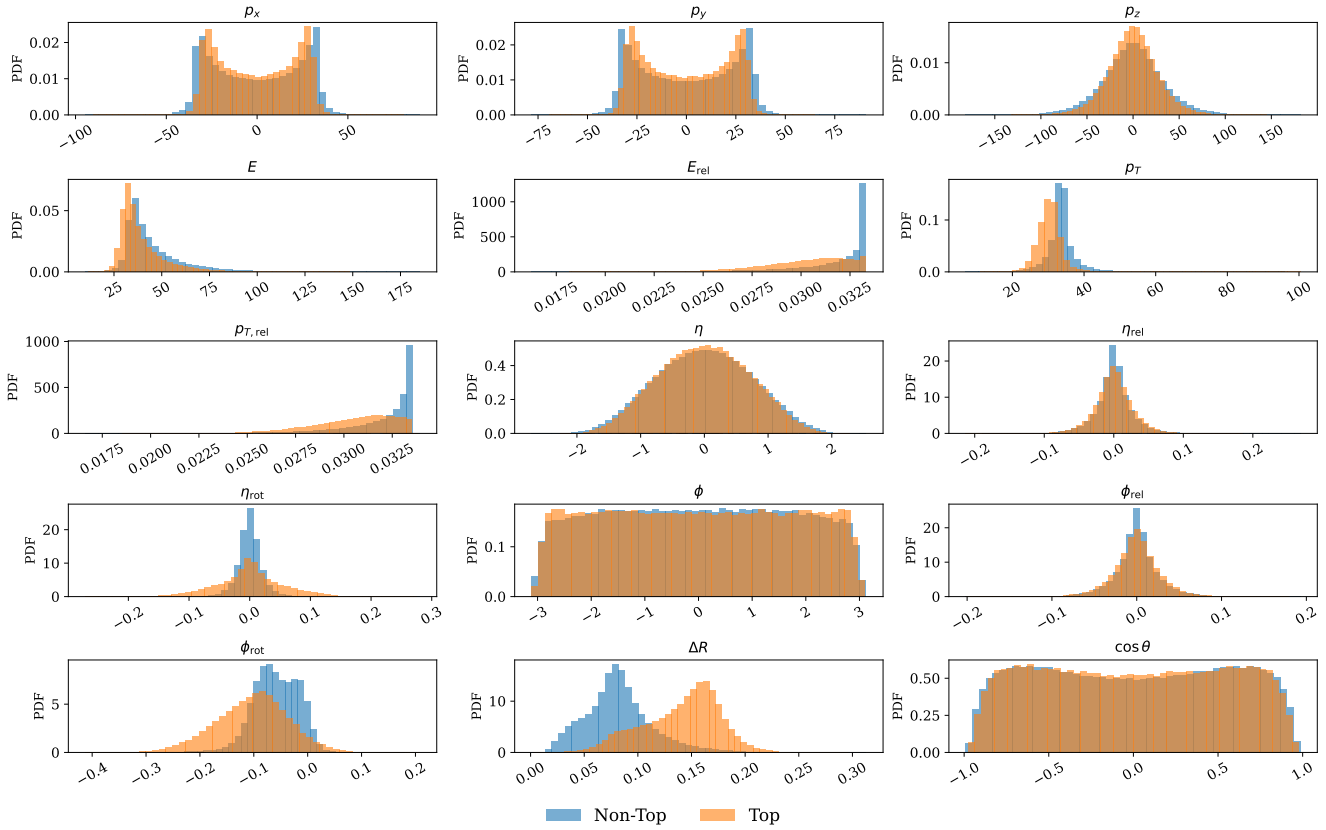

Fig. 1: Distributions of the input features, computed as average over all particles in each jet, for the training set.

Standard LSTMs have input i_t and forget f_t gates which are bounded in [0,1] due to sigmoid activation functions. This can make storage update irreversible, meaning that once information is largely forgotten it cannot be reintroduced. To empower xLSTM blocks with the ability to reopen or revise storage decisions, exponential gating is chosen (4) (5), which allows for more flexible ranges for the gates. Both blocks also introduce a normalizer $\mathbf{n_t}$ and a stabilizer $\mathbf{m_t}$ state to cope with potentially unbounded gates due to exponential gating. The normalizer state is defined accumulating the effect of past input and forget gates (12) (18) and ensures the hidden state does not explode when retrieved by rescaling it (13) (19). The stabilizer state instead avoids numerical instability from the exponential activations by keeping track of the maximum of input and forget log-gates (7), which is used to rescale i_t, f_t gates (8) (9) .

In a **mLSTM block**, the storage capacity of a conventional LSTM is increased by moving from the standard vector cell state to a richer *matrix memory* $\mathbf{C} \in \mathbb{R}^{d\times d}$, where d is the hidden dimension. At each timestep t (corresponding to a particle in the jet sequence), the input samples $\mathbf{x_t}$ are projected through linear layers (14) (15) (16) into three distinct $\mathbb{R}^d$ vectors: queries, $\mathbf{q_t}$, keys $\mathbf{k_t}$ and values $\mathbf{v_t}$. A key, value pair is therefore associated with each particle: $\mathbf{k_t}$ encodes a representation of an "address", defining where to store the corresponding value vector $\mathbf{v_t}$ (the "content" to store) within the memory matrix. Then at the following time $t+1$, the information in the memory cell relative to the given value $\mathbf{v_t}$ can be retrieved by the query vector $\mathbf{q_{t+1}}$ through matrix multiplication (19).
The memory state is updated at each timestep by an outer product of values and keys, weighted by a usual LSTM-like input gate i_t. In this way we write new content into memory in a location specified by $\mathbf{k}_t$. At the same time, a forget gate f_t controls the persistence of past memory exactly as in a LSTM. The memory update rule thus is given by (17), where we perform broadcasting since f_t, $i_t \in \mathbb{R}^d$.

In a **sLSTM block**, the conventional scalar memory cell c_t is retained. Together with the introduction of exponential gating and $\mathbf{n_t}$, $\mathbf{m_t}$ states, sLSTM allows multiple independent memory cells, each with its own gates, inside the same block. Each memory cell group, referred to as *head*, processes the same input sequence in parallel but maintains its own internal states ($\mathbf{c_t}$, $\mathbf{n_t}$, $\mathbf{m_t}$). Moreover, sLSTM introduces a novel form of *memory mixing*, by adding, for each memory cell, recurrent connections from the hidden state back to the cell input (10). This allows information stored in one cell to influence the gating of another cell within the same head, while no mixing between different heads is possible.

Results presented in [11] suggest that sLSTM blocks mainly contribute to improving learning stability with the normalization of the hidden states, whereas memory-rich mLSTM blocks enable the network to store and combine more complex patterns. As the original paper employs sequences with very few sLSTM blocks, we follow the same approach in this study.

B. *Forward pass equations.*

Common gates

$$\tilde{i}_t = W_i x_t + R_i h_{t-1} + b_i, \tag{1}$$
$$\tilde{f}_t = W_f x_t + R_f h_{t-1} + b_f, \tag{2}$$
$$\tilde{o}_t = W_o x_t + R_o h_{t-1} + b_o, \tag{3}$$
$$i_t = \exp(\tilde{i}_t), \tag{4}$$
$$f_t = \sigma(\tilde{f}_t) \text{ or } \exp(\tilde{f}_t), \tag{5}$$
$$o_t = \sigma(\tilde{o}_t), \tag{6}$$
$$m_t = \max(\log f_t + m_{t-1}, \log i_t), \tag{7}$$
$$i'_t = \exp(\log i_t - m_t), \tag{8}$$
$$f'_t = \exp(\log f_t + m_{t-1} - m_t). \tag{9}$$

sLSTM block

$$z_t = \tanh\left(W_z x_t + R_z h_{t-1} + b_z\right), \tag{10}$$
$$c_t = f'_t \odot c_{t-1} + i'_t \odot z_t, \tag{11}$$
$$n_t = f'_t \odot n_{t-1} + i'_t, \tag{12}$$
$$\tilde{h}_t = \frac{c_t}{n_t}, \qquad h_t = o_t \odot \tilde{h}_t. \tag{13}$$

mLSTM block

$$q_t = W_q x_t + b_q, \tag{14}$$
$$k_t = \frac{1}{\sqrt{d}} W_k x_t + b_k, \tag{15}$$
$$v_t = W_v x_t + b_v, \tag{16}$$
$$C_t = f'_t \odot C_{t-1} + i'_t \odot v_t k_t^\top, \tag{17}$$
$$n_t = f'_t \odot n_{t-1} + i'_t \odot k_t, \tag{18}$$
$$\tilde{h}_t = \frac{C_t q_t}{\max(|n_t^\top q_t|, 1)}, \qquad h_t = o_t \odot \tilde{h}_t. \tag{19}$$

V. Learning Framework

The xLSTM blocks are implemented from scratch in PyTorch. All model training was carried out on the CloudVeneto infrastructure using a virtual machine equipped with one NVIDIA T4 GPU, 15 vCPUs, and 90 GB of RAM.

We use binary cross-entropy loss with logits (`BCEWithLogitsLoss`), which is minimized using the Adam optimizer with an initial learning rate of $5 \cdot 10^{-3}$ (unless stated otherwise). In all cases, after training, the best model is selected as the one achieving the highest validation AUC. All models are initialized using Xavier-Glorot weight initialization. We adopt a batch size of 256 for LSTM models and 1024 for xLSTM models, primarily for computational reasons; we verified that smaller batch sizes do not improve performance.

A. *LSTM models*

We consider three, progressively more complex, LSTM-based architectures for comparison and benchmarking with the xLSTM models. A summary of the three models and hyperparameters is given in Tab. 2.

Vanilla LSTM: A single-layer LSTM with hidden size $d = 45$, chosen to match the dimension optimized in xLSTM blocks later. The LSTM output is pooled using a simple mean operation over the different particles and fed to a FFNN layer to perform classification (we have single output neuron because of the loss choice).

LSTM with attention: A single-layer LSTM, with hidden size $d = 45$, where we add a learnable attention layer. This allows the network to assign a different weight to each particle when computing the pooled representation for the FFNN layer. Additionally, we also introduce a weight k in the loss to penalize false negatives more heavily; here the weight is set as the ratio between the number of training samples of class 0 and class 1 ($k \approx 3.95$).

Multi-layer LSTM: A multi-layer LSTM with a final attention layer. Here we use Optuna to implement a grid search to optimize the number of layers $l \in \{1, 2, 3, 4\}$, hidden size $d \in \{8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48\}$ (identical for all layers), the penalization weight in the loss $k \in [1, 4]$ and the initial learning rate $\eta \in [10^{-5}, 10^{-2}]$.

TABLE 2: Summary of the LSTM models used for benchmarking. l denotes the number of layers, d the hidden size and k the loss weight penalizing false negatives.

Model	l	d	**Attention layer**	k
Vanilla LSTM	1	45	×	1
LSTM + Attention	1	45	✓	≈ 3.95
Multi-layer LSTM	4	48	✓	≈ 3.02

B. *xLSTM models*

We first implement a *single-mLSTM-block model*, with an attention layer to perform pooling before a final feed-forward dense classification layer. Following the design choices of [11], each mLSTM block applies a Layer Normalization to its inputs before projecting them to keys and values. We also reuse the optimal weight coefficient $k \approx 3.02$ previously tuned in the multi-layer LSTM to penalize false negatives in the loss. For this baseline model, we define a grid search to find the best value for the hidden dimension of the memory

matrix, $d \in \{10, 15, 20, 25, 30, 35, 40, 45, 50\}$ (preliminary trials with larger values showed no improvement). Highest validation AUC is achieved for $d = 45$, which is fixed in the following.

General xLSTM architecture. To explore richer architectures we design a general `JetTagger_xLSTM` model consisting of an arbitrary sequence of mLSTM and sLSTM blocks with hidden dimension $d = 45$. Residual (skip) connections are inserted between all blocks from the second layer onward (to match dimensions). Each block is followed by its own attention layer. The attention layer outputs from all blocks are concatenated and passed to a final dense layer for classification. In this setup, all blocks contribute directly to the final output and can learn separate representations that are not "washed out" by the other blocks. Fig. 2 provides a schematic representation of the architecture at the last timestep for the case of one mLSTM and one sLSTM block. This structure allows us to test with a grid search multiple block sequences in order to identify the configuration with the best performance. We consider architectures with up to 6 blocks, which have a comparable number of trainable parameters as the multi-layer LSTM. Training is performed for up to 70 epochs with early stopping after 10 epochs without improvement in validation AUC.

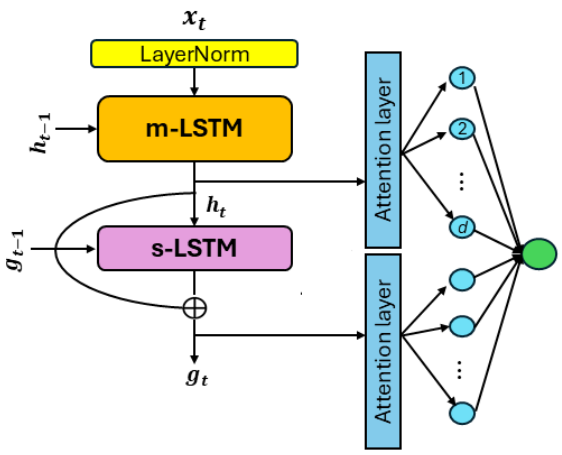

Fig. 2: Schematic representation of a JetTagger xLSTM model with one mLSTM and one sLSTM block at the final timestep.

VI. Results

In Tab. 3, we present the results on the test set for all models, with respect to the metrics of interest. For reference, we also show the training and validation loss curves for the multi-layer LSTM (Fig. 3) and for the best-performing xLSTM JetTagger model, given by the blocks sequence M-S-M, which records the highest validation AUC. (Fig. 4).

The performance results show that JetTagger xLSTM *do not* outperform standard LSTM-based architectures for this task. The best xLSTM model achieves comparable AUC and slightly worse rejection rates than the simpler Vanilla LSTM, while the multi-layer LSTM attains the best performance across all metrics of interest.

It is noteworthy that all multi-block xLSTM sequences achieve very similar AUC values, without showing a clear improvement trend, and that the specific block ordering has little impact. However, deeper architectures, in particular the M-M-S-M-M sequence, actually perform better in terms of background rejection on the test set compared to the best model chosen, as measured by $\text{Rej}_{30\%}$ and TPR@0.1%. This suggests that a potentially preferable approach could be to directly maximize *partial AUC* in the low-FPR regime (e.g. FPR $< 1\%$) when tuning hyperparameters and selecting the best model; this is indeed the region of greatest interest, and global AUC optimization does not appear sufficient to ensure optimal performance also in this region.

Since deeper and more complex architectures usually require very careful tuning of learning rates, initialization and regularization to optimize, it is possible that the xLSTM JetTagger model may not have reached its full potential, but results seem to suggest that simply stacking more blocks is unlikely to outperform multi-layer LSTM for this specific task. Moreover, the single mLSTM block model clearly underperforms single-layer LSTM with attention. This reinforces the idea that, for this dataset and binary classification task, the added architectural complexity does not seem to provide an advantage over standard LSTM.

These observations suggest some considerations. The xLSTM architecture is designed to capture long-term dependencies and multiple parallel representations through its block sequences and its internal key-query-value mechanism. This may be more advantageous for higher-dimensional samples (here we have a relatively small number of features) and longer input sequences. In light of this, it remains to be explored whether the xLSTM architecture can fully exploit its potential in more complex scenarios. In particular, further work needs to be carried out on bigger datasets containing jets with larger numbers of particles (e.g., up to 200 are usually used to define state of the art models [8]) and, in particular, on *multi-class jet tagging*, a problem of special relevance for low-level trigger systems. These tasks may allow the architecture to leverage separate representations and richer key-value interactions, potentially revealing advantages not evident in the current simpler setting.

VII. Concluding Remarks

In this study we consciously adopted a conservative evaluation context - a binary top-vs-background tagging task with a limited number of particles per jet - both to provide a controlled first test of the xLSTM architecture for jet tagging, and because limited computational resources made the fine-tuning of large models more challenging. This choice allowed for a direct comparison with standard LSTM

TABLE 3: Summary of test-set performance for LSTM models and xLSTM JetTagger variants with different block sequences. In bold the best models chosen according to validation AUC and the actual best performance for each metric.

Model	# Params	Val. AUC	Acc.	AUC	$\text{Rej}_{30\%}$	$\text{Rej}_{50\%}$	TPR@10%	TPR@1%	TPR@0.1%
Vanilla LSTM	11,206	0.9643	**0.9323**	0.9641	924.39	233.10	0.9141	0.6250	0.2886
LSTM + Attention	11,252	0.9653	0.9185	0.9652	1025.63	231.59	0.9159	0.6264	0.3026
Multi-layer LSTM	69,026	**0.9665**	0.9208	**0.9660**	**1170.56**	**253.69**	**0.9184**	**0.6372**	**0.3173**
xLSTM M	4442	0.9606	0.9134	0.9608	675.18	173.14	0.9072	0.5940	0.2477
xLSTM MS	20,913	0.9636	0.9216	0.9637	834.82	205.52	0.9138	0.6145	0.2776
xLSTM MM	17,043	0.9638	0.9199	0.9634	861.53	204.93	0.9140	0.6134	0.2836
xLSTM MMS	33,514	0.9638	0.9161	0.9638	791.85	210.54	0.9141	0.6194	0.2723
xLSTM MSM	33,514	**0.9644**	**0.9227**	0.9641	791.85	209.52	0.9148	0.6180	0.2739
xLSTM MMSM	46,115	0.9643	0.9176	0.9644	800.61	200.92	**0.9155**	0.6160	0.2741
xLSTM MMSMM	58,716	0.9641	0.9177	0.9643	**961.53**	**229.86**	0.9144	**0.6267**	**0.2941**
xLSTM MMSMMS	71,317	0.9639	0.9201	**0.9644**	924.39	204.35	0.9143	0.6151	0.2864
xLSTM MMSMMM	75,187	0.9639	0.9193	0.9639	924.39	209.11	0.9144	0.6121	0.2914

Fig. 3: Training and validation loss curves for the multi-layer LSTM model.

Fig. 4: Training and validation loss curves for the best xLSTM model: M-S-M sequence.

baselines but inevitably constrained the range of scenarios in which the architecture's advantages could emerge.

Within this setup, our results show that xLSTM JetTagger models underperform standard LSTM-based architectures in terms of AUC and rejection rates. Furthermore, all multi-block xLSTM sequences yield comparable AUC performances, indicating that merely further increasing depth or varying block order may not significantly improve results in this setting.

Future work should therefore investigate xLSTM on larger datasets with jets containing substantially more particles and, in particular, on multi-class jet-tagging tasks. We suggest adopting the maximization of partial AUC in the low-FPR range as objective. Such studies, coupled with more extensive hyper-parameter tuning on larger computational resources, will be essential to fully assess the potential of this new architecture in the context of jet tagging.

References

[1] L. D. Oliveira, M. Kagan, L. Mackey, B. Nachman, and A. Schwartzman, "Jet-Images-Deep Learning Edition," *JHEP*, vol. 07, no. 069, 2016. arXiv:1511.05190 [hep-ph].

[2] G. Kasieczka, T. Plehn, M. Russell, and T. Schell, "Deep-learning Top Taggers or The End of QCD?," *JHEP*, vol. 05, no. 006, 2017. arXiv:1701.08784 [hep-ph].

[3] S. Macaluso and D. Shih, "Pulling Out All the Tops with Computer Vision and Deep Learning," *JHEP*, vol. 10, no. 121, 2018. arXiv:1803.00107 [hep-ph].

[4] S. Choi, S. J. Lee, and M. Perelstein, "Infrared Safety of a Neural-Net Top Tagging Algorithm," *JHEP*, vol. 02, no. 132, 2019. arXiv:1806.01263 [hep-ph].

[5] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban, and D. Whiteson, "Jet Flavor Classification in High-Energy Physics with Deep Neural Networks," *Phys. Rev. D*, vol. 94, no. 112002, 2016. arXiv:1607.08633 [hep-ex].

[6] S. Egan, W. Fedorko, A. Lister, J. Pearkes, and C. Gay, "Long Short-Term Memory (LSTM) networks with jet constituents for boosted top tagging at the LHC," 2017. arXiv:1711.09059 [hep-ex].

[7] A. Collaboration, "Identification of Jets Containing b-Hadrons with Recurrent Neural Networks at the ATLAS Experiment," atl-phys-pub-2017-003, CERN, Geneva, Switzerland, 2017.

[8] H. Qu and L. Gouskos, "Jet Tagging via Particle Clouds," *Phys. Rev. D*, vol. 101, no. 050619, 2020. arXiv:1902.08570 [hep-ph].

[9] H. Qu, C. Li, and S. Qian, "Particle Transformer for Jet Tagging," in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, PMLR 162, pp. 18281–18292, 2024. arXiv:2202.03772 [hep-ph].

[10] J. Pearkes, W. Fedorko, A. Lister, and C. Gay, "Jet constituents for deep neural network based top quark tagging," 2017. arXiv:1704.02124.

[11] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, "xlstm: Extended long short-term," 2024. arXiv:2405.04517 [cs.LG].