DYNAMAX: Dynamic computing for Transformers and Mamba based architectures

Miguel Nogales[1], Matteo Gambella[2], and Manuel Roveri[2]
[1] Università della Svizzera Italiana, Lugano, Switzerland
[2] Politecnico di Milano, Milano, Italy
Email: miguel.nogales@usi.ch, {matteo.gambella, manuel.roveri}@polimi.it

***Abstract*—Early exits (EEs) offer a promising approach to reducing computational costs and latency by dynamically terminating inference once a satisfactory prediction confidence on a data sample is achieved. Although many works integrate EEs into encoder-only Transformers, their application to decoder-only architectures and, more importantly, Mamba models, a novel family of state-space architectures in the LLM realm, remains insufficiently explored. This work introduces DYNAMAX, the first framework to exploit the unique properties of Mamba architectures for early exit mechanisms. We not only integrate EEs into Mamba but also repurpose Mamba as an efficient EE classifier for both Mamba-based and transformer-based LLMs, showcasing its versatility. Our experiments employ the Mistral 7B transformer compared to the Codestral 7B Mamba model, using data sets such as TruthfulQA, CoQA, and TriviaQA to evaluate computational savings, accuracy, and consistency. The results highlight the adaptability of Mamba as a powerful EE classifier and its efficiency in balancing computational cost and performance quality across NLP tasks. By leveraging Mamba's inherent design for dynamic processing, we open pathways for scalable and efficient inference in embedded applications and resource-constrained environments. This study underscores the transformative potential of Mamba in redefining dynamic computing paradigms for LLMs.**

***Index Terms*—Large Language Models (LLMs), Dynamic Computing, Early Exit Neural Networks, Transformers, State-Space models**

I. Introduction

Natural Language Processing (NLP) has experienced transformative growth due to advancements in neural network architectures, particularly decoder-only models that leverage autoregressive processes for text generation [1], [2]. At the core of these innovations lie Transformer-based architectures, which utilize self-attention mechanisms to enable large-scale parallel processing [3]. This paradigm shift has substantially enhanced capabilities in language understanding and generation, setting new benchmarks for generalization and performance in NLP tasks (and many others).

Decoder-only models have emerged as key contributors to this progress. Open-source frameworks like Llama [4] and Mistral [5], alongside proprietary systems such as GPT [6] and Claude [7], demonstrate the effectiveness of pre-training on large-scale corpora followed by task-specific fine-tuning. This transfer learning approach enables these models to generalize across domains, delivering state-of-the-art results in text generation, summarization, and other language-related applications. Despite these advancements, the computational and energy demands of Transformers remain a significant challenge, especially since their model size is not only steadily growing, but also the inference-time computation needs have increased due to the use of Test Time Compute (TTC) [8], where models such as closed OpenAI's o1 [9] or open source DeepSeek's R1 [10] make use of it to enhance performance. To address this issue, this work explores technical innovations aimed at improving efficiency while maintaining high performance. Architectures such as Mamba [11], which integrate state-space models, offer a novel solution to mitigate the scalability limitations inherent to standard Transformers. Furthermore, emerging techniques such as early exit (EE) mechanisms [12], widely used in computer vision [13], [14], are being adapted to NLP [15]. By embedding auxiliary classifiers within intermediate layers, these mechanisms allow models to terminate processing early when confidence thresholds are met, reducing computational overhead and energy consumption.

To the best of our knowledge, there is no prior study on the performance of Mamba with EEs, nor EEs classifiers based on Mamba architectures.

In this paper, we present DYNAMAX, a framework of dynamic computing for decoder-only Transformers and Mamba-based architectures with EEs aiming at improving the trade-off between the performance and computational cost of these LLMs. Early exit classifiers are added to the latter half of the LLM and trained by knowledge distillation from the full model and applying a relaxation to avoid training instability. The inference follows a general token forwarding scheme with EEs where it is possible to stop computation when enough confidence is achieved. In Transformers, a simple tweak in how missing states are handled permits more enhancement of the computational efficiency. Three evaluation datasets have been selected to assess the effectiveness of the framework on diverse linguistic tasks. In particular, we compared the efficacy in increasing computational efficiency of EE techniques with respect to another method named layer pruning, which showed to be promising for LLMs [16].

The innovations presented in this work are the following:

- The addition of EEs to Mamba architectures.
- The use of Mamba as an EE classifier exploiting its unique properties.
- An alternative way of training the EE classifiers.
- A different, more efficient, way to deal with missing states in Decoder-only Transformers.

The work is structured as follows: following the Introduction, section III will cover technicalities about the models' architectures, early exits, and current implementations. Section IV will cover the implementations of the innovations presented in the work. Lastly, the results of the conducted experimental campaign will be presented, showing how computation savings affect performance. To facilitate comparisons and reproducibility, the source code of DYNAMAX is released to the scientific community as a public repository. [1]

II. Related literature

The usage of EEs in Transformers has been widely addressed with encoder-only models (such as BERT [17]), where they have found great success. Some examples of this are [18] or [19], where EEs accelerate the costly inference of those models. Additionally, the work [20] satisfactorily applies EEs to Transformers in graph neural networks for additional speed-up gains. There are also examples of Transformers with EEs in vision applications such as [21] or [22]. In the field of Transformers which include decoders with EEs attached, the works [23] and [24] are found, which set the basis of EE-accelerated Transformer-based models. Their performance, while modest, opens up a new framework for the inference of LLMs. Lastly, focusing on decoder-only models, the works [25] and [26] found great speed-ups by using draft models to generate the text. These approaches focus on the speed-up aspect of EEs, but do not tackle the computational complexity or energy spent.

Key works on decoder models with EEs include [24] and [23], where the latter introduced the concept and CALM refined its framework. EEs are applied across all model blocks, making token-wise exit decisions based on entropy, state saturation, and neural network-based predictions. Entropy-based exits offer high confidence but incur high computational costs due to Softmax operations. State saturation, which measures hidden state differences, is simpler but less effective. Neural network predictors provide a balanced trade-off between efficiency and accuracy. Unlike other EE methods, these frameworks share a common output classifier across all exits.

There are latter works such as [25], which introduce the FREE framework, demonstrating competitive results against CALM. Like prior research, they utilized T5 family models [27], employing a dual-model approach where a shallow model partially computes tokens via early exits, while a deep model processes them fully. Their key contribution is an innovative decoding strategy: a new cache mechanism retains early-exited tokens, enabling parallel computation of their key values when full processing is required.

Similarly, the work by Chen et al. [26] explores a comprehensive framework for training and inference of LLMs with EE. Achieving speed-ups of up to $\times 3$, their approach leverages extensive parallelism, including pipeline parallelism during decoding. By fully utilizing GPU resources, they enable the use of entire encoder transformer models as early exit classifiers, significantly enhancing efficiency.

In this work, from the three main methods to assign a confidence measure to exit in decoders with EEs, the neural network-based one criterion is selected. This method is more suited to fulfill the requirements imposed for constrained environments, while maintaining performance.

III. Technical background

A. Transformer and Mamba Architectures

The Transformer [3] and Mamba [11] model architectures represent two distinct approaches to handling long-range dependencies in sequential data. Transformers leverage self-attention to capture global context within sequences by computing pairwise attention across all tokens. Self-attention computes a weighted sum of values (V), where the weights are determined by the compatibility between the query (Q) and key (K) vectors. Mathematically:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where d_k is the dimension of the key vectors. The multi-head attention mechanism extends this by computing multiple attention heads in parallel:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

where each attention head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

Transformer models offer powerful parallelization and expressive capabilities but are computationally and memory-intensive, with training costs scaling quadratically with sequence length. To enhance efficiency, transfer learning has become a key strategy in NLP, enabling models to be pre-trained on large datasets and fine-tuned for specific tasks. While pretraining is resource-intensive, parameter-efficient fine-tuning techniques (PEFT) [28], such as LoRA (low-rank adaptation) [29] or adapters, reduce computational demands, allowing models to leverage open-source pre-trained weights for adaptation across domains.

The main focus of this work is the reduction of computational complexity; the number of operations of the Transformer block is re-examined. The main source of computational cost of the block is the self-attention mechanism and the feedforward network. Note that we will not take into account the layer norm or the residual connections' compute.

$$\begin{aligned}\text{Total Operations} &= \text{Total}_{\text{MHSA}} + \text{Total}_{\text{FFN}} \\ &= \left[8 \times T \times d_{\text{model}}^2 + 4 \times T^2 \times d_{\text{model}}\right] \\ &\quad + \left[16 \times T \times d_{\text{model}}^2\right] \\ &= \left[24 \times T \times d_{\text{model}}^2 + 4 \times T^2 \times d_{\text{model}}\right]\end{aligned} \tag{1}$$

T represents the sequence length and d_{model} represents the internal dimension of the Transformer. We can see how attention contributes with quadratic dependence on the length. The

[1] https://github.com/Xigm/DYNAMAX

complexity of inference implementing one token forwarding with KV caching would be very similar, but dependence on the sequence length is decreased in all terms by one order of magnitude.

However, Mamba is an efficient architecture that utilizes state-space models (SSMs) for sequence processing. Unlike Transformers, which rely on self-attention, Mamba processes sequences using linear operations in combination with learned dynamics. SSMs describe the evolution of a hidden state x_t over time based on input u_t:

$$x_{t+1} = Ax_t + Bu_t, \quad y_t = Cx_t + Du_t,$$

where A, B, C, and D are learnable parameters, x_t is the state vector, u_t is the input, and y_t is the output.

The convolutional form of SSMs allows for efficient sequence processing:

$$y = \text{SSM}(u) = C * (K * u) + Du,$$

where K is a kernel derived from the state-space dynamics.

Mamba exploits SSMs to capture input-dependent information, trading some parallelization for scalability in long-sequence tasks. To address this, a hardware-aware algorithm ensures efficient training. Mamba 2 [30] introduces architectural optimizations, enhancing training flexibility and connecting SSMs with linear attention techniques. Unlike self-attention, which explicitly computes global dependencies, Mamba captures them implicitly through learned dynamics with a constant-sized state.

In a similar way as the Transformer, the computational complexity can be accurately approximated by the cost of its projection matrices (the input and output projections of the Mamba block) because their other components, such as the one-dimensional convolution and the SSM block, are at least an order of magnitude smaller.

$$\begin{aligned}\text{Total Operations} &= \text{Total}_{\text{input}} + \text{Total}_{\text{output}} \\ &= \left[2 \times d_{\text{model}}^2 + 2 \times d_{\text{model}}^2\right] \\ &\quad + \left[2 \times n_{\text{groups}} \times d_{\text{state}} \times d_{\text{model}}\right] \\ &\quad + \left[2 \times d_{\text{model}}^2\right] \\ &= \left[6 \times d_{\text{model}}^2 + 2 \times n_{\text{groups}} \times d_{\text{state}} \times d_{\text{model}}\right]\end{aligned} \tag{2}$$

The first term of Equation 2 is the projection of input data, precisely the computation of matrices, the projection of the input and computation of matrices B and C. The output term is just the output projection of the computed state.

B. Early Exit Mechanisms

Early exit mechanisms (EE) [12] enhance efficiency by halting computations early when confidence thresholds are met, reducing resource usage and processing time. This is particularly beneficial for large models like Transformers, where increasing sequence length amplifies computational demands, enabling faster predictions without compromising performance. In real-time applications, EEs dynamically allocate resources based on data complexity, which is advantageous for models like SSMs that benefit from linear complexity.

EE mechanisms function via checkpoints at various model stages, assessing confidence in intermediate predictions and terminating computation when thresholds are met. Typically, checkpoints are positioned after each block in architectures like Transformers or SSM-based models (e.g., Mamba), allowing precise resource management. Training methods such as joint training [31], layer-wise training [32], and knowledge distillation [33] enhance intermediate layer performance, enabling confident early predictions.

Another important implication of EEs is that they are orthogonal to any other efficiency-related modification of NNs. The most important techniques in this category are pruning [34], which removes redundant weights or neurons to improve efficiency with minimal performance loss, quantization [35], which reduces the precision of the weights (e.g. from Float16 to Float8) and efficient attention techniques, such as works [36], [37], where they modify the attention mechanism to reduce its complexity, whether it's by linearizing the operation (removing the softmax operation) or by using windowed attention. The only one which is not that suitable is batching, the processing of different data samples in parallel, which even though the computation reduction is achieved, does not receive the benefits of reduced delay. In the context of this work, layer pruning [16] is particulary relevant, as it allows for direct comparison with EEs. Both techniques make the model use a smaller number of layers than the original one, the main difference being that EE performs the allocation of computation on a per-token basis. Instead, layer pruning assigns a static amount of block for all tokens. This allows reducing memory footprint, unlike EEs.

IV. Proposed solution and implementation

In this section, we present the technical decisions and approaches used in the implementation of early exit mechanisms in LLMs. This section details the selection of models and datasets, as well as the training and inference strategies for implementing early exits in Transformer and Mamba architectures.

A. Models

Pre-trained weights for Mistral 7B v0.3 [5] and Codestral Mamba 7B [38], similar in model size, available in the Hugging Face repository [39], were used for this study. Mistral 7B is a Transformer, set with pre-trained weights, which integrates its architecture with sliding window and grouped query attention to optimize computational cost, while Codestral Mamba 7B, specifically optimized for code tasks, employs Mamba 2 blocks to improve efficiency in complex tasks. These models serve as a baseline to analyze the impact of early exits on computational efficiency and overall performance.

B. Early Exits for Computational Efficiency in Autoregressive Generation

Early exits have been implemented to reduce the computational cost of autoregressive generation by enabling interme-

Algorithm 1 Token Forwarding with Early Exits during Inference in Large Language Models

Input: Token T, Early Exit Threshold θ
Output: Final output $output$
$x \leftarrow$ tokenize(T)
$pe \leftarrow$ positional_encoding(x)
$z \leftarrow x + pe$
for each transformer block b in the model backbone **do**
  $z \leftarrow b(z)$
  if b contains an early exit classifier **then**
    $exit\_output \leftarrow$ early_exit(z)
    if $exit\_output \geq \theta$ **then**
      for each subsequent layer b' in the remaining model backbone **do**
        $states \leftarrow$ partial_forward(z)
      end for
      break
    end if
  end if
end for
$z \leftarrow$ last_block(z)
$z \leftarrow$ norm(z)
$output \leftarrow$ head(z)

return $output$

diate exits based on model confidence. To minimize overhead, neural network classifiers are integrated at specific layers in the latter portion of the models, as early exits at these positions are more likely to yield accurate results without disrupting token dependencies in the generation process. These EE classifiers are designed in three distinct manners:

- **CALM Style**: A simple one-layer feed-forward network, mimicking the ones used with encoder-decoder Transformers in [24].
- **Transformer Block-Based Network (FFN)**: An feed-forward network similar to those in transformer blocks, taking inputs sized to the backbone's internal dimension and expanding to four times the size at the second layer, and then reducing the size to two, to flag the exit.
- **Mamba Block**: In a similar way as the prior, the Mamba block is used, modifying the output projection layer to output two values instead of the whole internal-dimension-sized hidden state.

The rationale for employing the Mamba block is its ability to account for state dependencies, which FFNs lack without compromising computational cost, which is the downside of the Transformer block. Transformer blocks are not utilized as EE classifiers due to their complexity and the associated increase in computational cost, particularly regarding the growing size of their key-value cache during forwarding.

Each EE classifier outputs two values, processed through a Softmax function, to decide whether to exit at the current layer or continue to the next. This decision is guided by a pre-defined confidence threshold, which determines the computational budget. Higher thresholds require greater confidence from the classifier to exit early, thereby reducing computational cost less than lower thresholds but maintaining the original performance.

The classifiers are connected to the backbone by redirecting their hidden states, after applying layer normalization, to the network. In the case of the Mamba backbone, taking into account that it has a constant size state, it was considered also to include as input its state, or a projection of it in the case of the convolutional block, but it was discarded due to growth in parameter size and not much improvement in prediction quality. When using Mamba as EE classifier, the general strategy is maintained but removing the typical skip connection of the block.

C. Training Early Exit Classifiers

The training of EE classifiers follows a knowledge distillation approach, where the early exit outputs are compared to the full model output, referred to as the *oracle*. Classifier inputs consist of the hidden values of the token being forwarded. Training is supervised using the cross-entropy loss, optimizing classifier accuracy against the oracle output.

To address the challenge of unbalanced targets during training, caused by the strict requirement for EE outputs to match the oracle exactly, this work proposes a relaxation inspired by the top-k sampling process. Exits are triggered if the most probable EE token is among the top-k most probable tokens predicted by the oracle, being equivalent to the CALM's oracle setting with $k = 1$. This approach tolerates minor variations in output probabilities and enhances training stability.

Training is parallelized across classifiers. In the case of the Mamba EE classifier, we use the parallel form of the SSM to perform this training. When jointly training them, the overall loss is adjusted using linearly decaying weights based on each classifier's position within the network.

D. Inference with Early Exits

The inference process for EEs differs between Transformer and Mamba architectures due to their structural variations. In Transformers, KV caching is used to store intermediate representations essential for autoregressive generation. When an EE is triggered, the KV cache of subsequent Transformer blocks lacks representations for exited tokens. This issue is traditionally addressed by partially forwarding the token through the remaining blocks to update their states. However, this work proposes a more computationally efficient alternative: copying the cached values directly to subsequent blocks, bypassing the need for recomputation. This way, for each Transformer block $block_n$ which is not activated for the forwarding of the current token, will have its KV cache updated with the keys and value from the

In the Mamba architecture, the recurrence-based design allows for incremental updates to internal states. When an early exit is triggered, state updates are managed similarly to the partial forward method in Transformers. Alternatively, skipped

recomputation can be employed, leaving the states unchanged, under the assumption that a posterior token forwarded through the entire model will eventually update all states with any missing information. A repetition penalty is introduced to mitigate token repetition during generation. In the case of Mamba as EE classifier, also the recurrent mode is employed to allow for constant complexity, updating in each forward the classifier state. No recomputation of states is used when using Mamba as EE classifier.

The overall inference process for early exits, generalized to both Transformers and Mamba architectures, is described in Algorithm 1. This algorithm outlines the steps for token forwarding during inference, where tokens are sequentially processed through layers. At each layer, the EE condition is evaluated. If met, the loop terminates, reducing computational cost. The shared model head subsequently generates the final output, ensuring an efficient and lightweight early exit approach.

V. Results

This section presents the experimental results and a discussion of the effectiveness of EEs versus layer pruning in LLMs. The first part addresses model performance across selected evaluation tasks, focusing on the Mamba and Transformer models with various EE configurations and with and without recomputation. The computational reduction factor used to evaluate the models is the relationship between the total number of operations of using the whole model versus the computation actually used by the model. In the case of layer pruning, it is directly related to the number of enabled blocks. In the case of EEs, the complexity, aside from the number of backbone blocks activated, has two additional factors. First, the evaluation of the EEs, which, depending on the architecture, may contribute to the overall computation; in the case of the CALM classifiers, it is negligible, while in the case of Mamba and the FFN, it will contribute. The other source of computational cost is the recomputation of states. This process adds to the total expenditure a fraction of the cost of the entire block (partial forward). In the case of Transformers, this process consists of the computation of the KV cache with the current token, which costs $\frac{1}{6}$ of the whole block, while in Mamba it is the update of the 1-dimensional convolutional layer and the SSM for a cost of $\frac{9}{26}$. Lastly, performance is evaluated by sweeping the values of confidence thresholds. The same value is used for each classifier in a single test, to reduce the search space, as combinations of these would have resulted in too many different evaluations. Sometimes, when the threshold is set too low, the model proceeds to output inconsistent responses that are composed of the same repeated token. If this event occurs a certain number of times, that configuration for the evaluation is not considered valid and then not shown in the graphs.

A. Datasets for training and benchmarking

We use multiple datasets for training and evaluation. For training, the FineWeb-Edu dataset [40] is used, comprising over one billion curated paragraphs. Only a sample of 10 billion tokens was used, focusing on representative examples that avoid repetitive patterns and enhance model training efficiency. Three evaluation datasets have been chosen to assess different language understanding tasks: TruthfulQA [41], CoQA [42] and TriviaQA [43]. A crucial aspect of these datasets is that they evaluate text generation, while also accounting for knowledge. TriviaQA is a test which leverages both generation and knowledge, but many of them consist of single-word answers. For that reason, a subset of one thousand samples was selected from the ones with the longer answers. In this way, the knowledge-based aspect of the test is achieved while improving the text generation part. The metric used to evaluate this test is exact match. CoQA emphasizes conversational question-answering, testing the ability of the model to maintain context. Its main metric is also an exact match, but F1 is also provided. Lastly, TruthfulQA for text generation is used, evaluated with BLEU and Rouge metrics. This set of datasets provides a comprehensive evaluation framework for assessing the effectiveness of early exit in various linguistic tasks.

B. Performance in the Tasks

We used procedures available from Eleuther AI for the evaluation harness [44]. In the following figures, we compare the performance of early exits versus layer pruning techniques, using task-specific metrics: exact match for TriviaQA, exact match and F1 for CoQA, and Bleu and Rouge for TruthfulQA. It is worth noting that these metrics, specially the ones for TriviaQA and CoQA, provide a strict evaluation for both text generation and knowledge performance of the models. TriviaQA is evaluated with two shots, while TruthfulQA is evaluated with one. The upper right corner of each plot represents optimal performance, achieving high accuracy with maximum computational savings. Typically, model performance decreases as computational resources are reduced, moving from the top left to the bottom right of each figure. In these experiments, a configuration of four early exits is used, placed in the second half of the model's backbone. Layer pruning is implemented by disabling from the Nth - 1 block, as many blocks as desired, following work [16]. No afterwards healing of the backbone is performed, considering both the inclusion of EE's and layer pruning an adhoc method.

1) TriviaQA: Figure 1 presents the performance results for the TriviaQA task, comparing early exits and layer pruning for the Transformer and Mamba models. Early exits generally outperform layer pruning by achieving a better balance between computational savings and model accuracy. This occurrence is likely to be happening due to having a big information loss when using layer pruning, showing early exits' largest strength, selective usage of computation. The effectiveness of EE is particularly evident at lower confidence thresholds, where gains are modest (around $1.2\times$) but still result in higher accuracy than at baseline. In contrast, layer pruning causes noticeable performance drops, particularly in knowledge-focused tasks such as TriviaQA, where the model does not rely on external information.

The recomputation or not of states appears to have a limited impact on performance improvement, but diverges noticeably between the two architectures. In the Transformer model, the recomputation does not affect too much total computation spent, but it shows that it may allow for more consistent results in some cases, where lower confidence thresholds can be set. The case of Mamba is very different, as it shows a bigger difference between recomputation or not. This event seems to be linked to the higher cost of recomputing the states with the Mamba-based backbone, and it is consistent with the different classifiers.

Between the three configurations of the EE classifier, there is a big difference between the CALM-style classifiers and the other two. These classifiers seem not enough for this problem, compared to their original application. Between the other two, even though of similar parameter sizes, Mamba shows as a better model for EE prediction, possibly because Mamba provides a state, which can keep information of the currently generated text and update it as it goes, which does not happen with the FFN. It also allows for a higher degree of freedom to set the confidence threshold, with a larger spawn over the x axis for the Transformer.

When comparing both model backbones, the EEs provide similar performance gains relative to the baseline, with a consistent degradation per computational cost reduction.

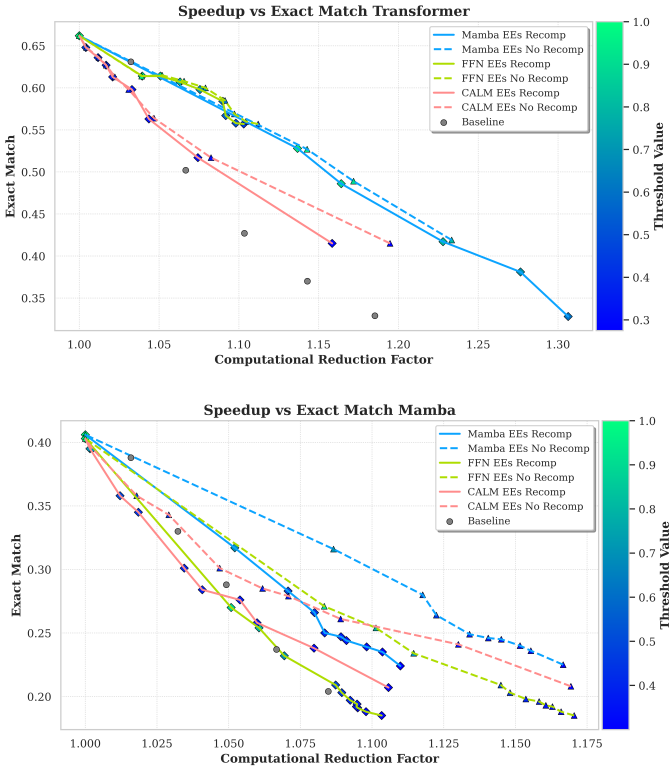

Fig. 1. Comparison of EE and layer pruning performances in the TriviaQA set for Mistral and Mamba 7B models.

2) CoQA: The results on CoQA, displayed in the Figure 2 for Transformer and in Figure 3 for the Mamba based architecture, reveal a similar trend to TriviaQA, but with layer pruning being more interesting overall. The Transformer works great in general with the EE, while Mamba does not reach that level of performance with the exits, showing that this pruned configuration works well.

Again, not recomputing the states does not show a significant improvement over recomputing them in the Transformer case, unlike Mamba which shows a big difference. The behaviours of the models both considering Exact match or F1 as a metric are very similar, but the difference in F1 is a bit greater in the case of the recomputation vs no recomputation. Exact match might be a bit strict, so having a better performance with the F1 metric means that even though the model is not outputting exactly the same tokens, they still keep a similar meaning.

The trend of the CALM classifiers is very similar to the one presented by the layer pruning and seems to be the opposite of the FFN. However, the latter is still better computationally. Using a Mamba based classifier seems best mainly thanks to the greater range of confidence thresholds that are available.

However, for higher computational savings, EE configurations provide better performance. In particular, the CoQA task benefits more from not recomputing states, especially as shown by the F1 metric, which indicates that recomputation may introduce unnecessary overhead without proportionate gains in performance.

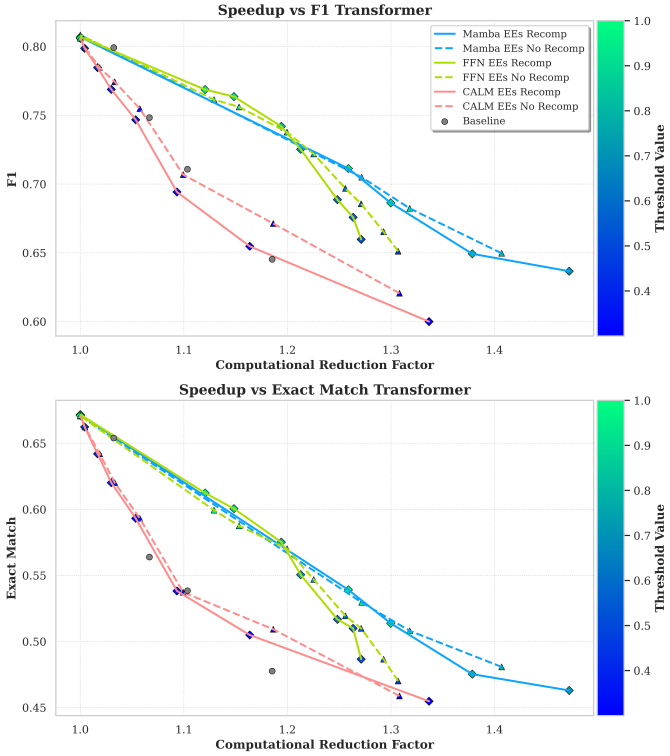

Fig. 2. Performance of EE and layer pruning in the CoQA set for Mistral 7B.

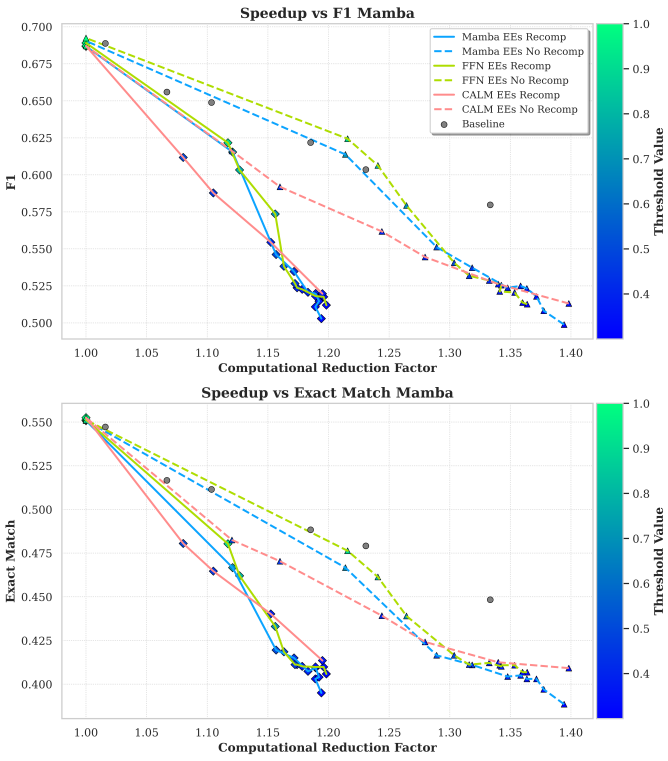

Fig. 3. Performance of EE and layer pruning in the CoQA set for Mamba.

3) TruthfulQA generation: The performance for the TruthfulQA generation task, shown in Figures 4 and 5 presents a more complex behavior than the one presented in the previous tasks, probably because this task is more focused on the text generation aspect than on the knowledge part, both due to the task itself and the metrics that are used to account for its performance.

For example, in general, the EE configuration is more interesting than the layer pruning strategy, mostly with the Transformer, and slightly less so with the Mamba model. The latter presents a big drop in performance while in Transformer the performance is kept or even improved. Improvements in performance are likely due to a combination of a maintenance in the possible answers of the model, and at the same time a drop in the wrong answers.

Analyzing the performance of the individual EE configurations, the Mamba based configuration shows a great improvement in performance over the other two, achieving a great computational reduction factor with no performance loss in the task and with a larger range of freedom compared to the other alternatives.

This task seems to be more suited for the analysis of the Transformer model, which excels. This could be partially due to the set of weights selected for the tests.

For this analysis, we used the *acc* submetric of TruthfulQA, which exhibited the most interesting performance patterns, among the three submetrics. Unlike exact match metrics, which are rigid, Bleu and Rouge offer a more flexible evaluation that captures nuanced differences in generated text.

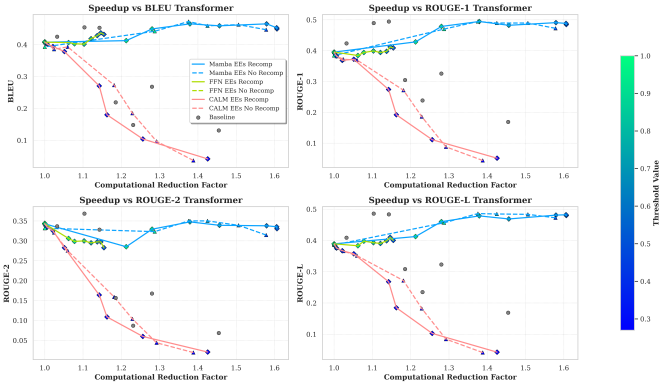

Fig. 4. Performance of EE and layer pruning in the Truthful QA generation (acc) set for Mistral 7B.

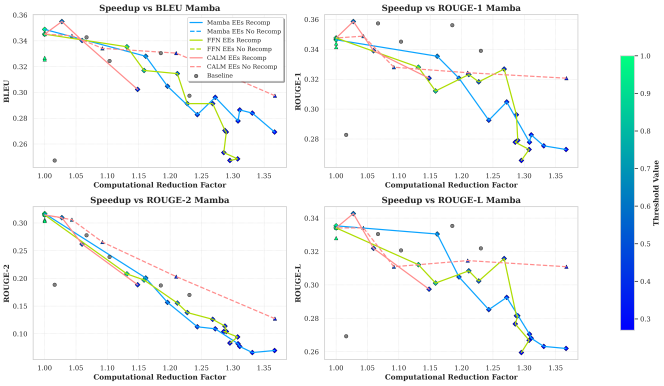

Fig. 5. Performance of EE and layer pruning in the Truthful QA generation (acc) set for Mamba.

VI. Conclusions and Future Work

This work investigated early exit mechanisms (EEs) in large language models (LLMs), specifically within the Transformer and Mamba architectures, to improve computational efficiency and inference speed. By implementing EEs in models such as Mistral 7B and Codestral 7B, we assessed the impact of dynamic computation on performance, accuracy, and energy consumption in NLP tasks, using datasets like TruthfulQA, TriviaQA, and CoQA. The results indicate that EEs effectively reduce inference time in low-latency applications, showing a good accuracy-inference time trade-off. In particular, the addition of Mamba-based EE classifiers showed to add more resilience to performance degradation, leveraging their state-space structure to complement early exits, thus supporting memory-efficient processing of long-range dependencies and with a constant inference cost.

Although EEs offer promising enhancements for efficient NLP deployment, they require careful tuning to balance computational savings with model fidelity, particularly in high-accuracy scenarios. Future work could refine EE strategies through adaptive thresholds that respond dynamically to context, latency, or device constraints. Combining EE with optimizations such as pruning and knowledge distillation

could further improve performance for resource-constrained applications. Additionally, applying EEs in pretraining, or integrating parameter-efficient fine-tuning methods, may foster confidence-aware models that perform effectively across diverse, high-demand tasks, such as medical diagnostics and autonomous systems, where transparency and efficiency are crucial.

REFERENCES

[1] L. Yu, B. Shi, R. Pasunuru, B. Muller, O. Golovneva, T. Wang, A. Babu, B. Tang, B. Karrer, S. Sheynin, C. Ross, A. Polyak, R. Howes, V. Sharma, P. Xu, H. Tamoyan, O. Ashual, U. Singer, S.-W. Li, S. Zhang, R. James, G. Ghosh, Y. Taigman, M. Fazel-Zarandi, A. Celikyilmaz, L. Zettlemoyer, and A. Aghajanyan, "Scaling Autoregressive Multi-Modal Models: Pretraining and Instruction Tuning," *arXiv*, Sept. 2023.

[2] B. Wang, W. Ping, P. Xu, L. McAfee, Z. Liu, M. Shoeybi, Y. Dong, O. Kuchaiev, B. Li, C. Xiao, A. Anandkumar, and B. Catanzaro, "Shall We Pretrain Autoregressive Language Models with Retrieval? A Comprehensive Study," *ACL Anthology*, pp. 7763–7786, Dec. 2023.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and Efficient Foundation Language Models," *arXiv*, Feb. 2023.

[5] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7B," *arXiv*, Oct. 2023.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[7] Anthropic, "Claude 3 model card," tech. rep., Anthropic Inc., 2023. 2024-03-04.

[8] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters," *arXiv*, Aug. 2024.

[9] OpenAI, "Learning to reason with LLMs," Sept. 2024.

[10] D.-A. I., D. Guo, D. Yang, and e. A. Zhang, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," *arXiv*, Jan. 2025.

[11] A. Gu and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," *arXiv*, Dec. 2023.

[12] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt, "Early exit optimizations for additive machine learned ranking systems," in *Proceedings of the third ACM international conference on Web search and data mining*, pp. 411–420, 2010.

[13] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469, IEEE, 2016.

[14] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning Dynamic Routing in Convolutional Networks," *arXiv*, Nov. 2017.

[15] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "Dynabert: Dynamic bert with adaptive width and depth," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9782–9793, 2020.

[16] A. Gromov, K. Tirumala, H. Shapourian, P. Glorioso, and D. A. Roberts, "The Unreasonable Ineffectiveness of the Deeper Layers," *arXiv*, Mar. 2024.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv*, Oct. 2018.

[18] H. Yin, A. Vahdat, J. Alvarez, A. Mallya, J. Kautz, and P. Molchanov, "AdaViT: Adaptive Tokens for Efficient Vision Transformer," *arXiv*, Dec. 2021.

[19] G. Xu, J. Hao, L. Shen, H. Hu, Y. Luo, H. Lin, and J. Shen, "Lgvit: Dynamic early exiting for accelerating vision transformer," in *Proceedings of the 31st ACM International Conference on Multimedia*, pp. 9103–9114, 2023.

[20] Q. Wu, W. Zhao, Z. Li, D. P. Wipf, and J. Yan, "Nodeformer: A scalable graph structure learning transformer for node classification," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27387–27401, 2022.

[21] L. Li, Y. Lin, D. Chen, S. Ren, P. Li, J. Zhou, and X. Sun, "CascadeBERT: Accelerating Inference of Pre-trained Language Models via Calibrated Complete Models Cascade," *arXiv*, Dec. 2020.

[22] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju, "FastBERT: a Self-distilling BERT with Adaptive Inference Time," *arXiv*, Apr. 2020.

[23] M. Elbayad, J. Gu, E. Grave, and M. Auli, "Depth-Adaptive Transformer," *arXiv*, Oct. 2019.

[24] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Tran, Y. Tay, and D. Metzler, "Confident adaptive language modeling," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17456–17472, 2022.

[25] S. Bae, J. Ko, H. Song, and S.-Y. Yun, "Fast and Robust Early-Exiting Framework for Autoregressive Language Models with Synchronized Parallel Decodingt5," *arXiv*, Oct. 2023.

[26] Y. Chen, X. Pan, Y. Li, B. Ding, and J. Zhou, "EE-LLM: Large-Scale Training and Inference of Early-Exit Large Language Models with 3D Parallelism," *arXiv*, Dec. 2023.

[27] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[28] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey," *arXiv*, Mar. 2024.

[29] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv*, June 2021.

[30] T. Dao and A. Gu, "Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality," in *International Conference on Machine Learning (ICML)*, 2024.

[31] M. Gambella and M. Roveri, "Edanas: Adaptive neural architecture search for early exit neural networks," in *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2023.

[32] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems* (B. Schölkopf, J. Platt, and T. Hoffman, eds.), vol. 19, MIT Press, 2006.

[33] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv*, Mar. 2015.

[34] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[35] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits," *arXiv*, Feb. 2024.

[36] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-Attention with Linear Complexity," *arXiv*, June 2020.

[37] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," *arXiv*, Apr. 2020.

[38] Mistral AI Team, "Codestral mamba," 2024. 2024-07-16.

[39] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," *arXiv*, Oct. 2019.

[40] A. Lozhkov, L. Ben Allal, L. von Werra, and T. Wolf, "Fineweb-edu," May 2024.

[41] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring How Models Mimic Human Falsehoods," *arXiv*, Sept. 2021.

[42] S. Reddy, D. Chen, and C. D. Manning, "Coqa: A conversational question answering challenge," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 249–266, 2019.

[43] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension," *arXiv*, May 2017.

[44] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 07 2024.