HELIOS : ADAPTIVE MODEL AND EARLY-EXIT SELECTION FOR EFFICIENT LLM INFERENCE SERVING

Avinash Kumar [1] **Shashank Nag** [1] **Jason Clemons** [2] **Lizy John** [1] **Poulami Das** [1]

ABSTRACT

Early-Exit Large Language Models (EE-LLMs) enable high throughput inference by allowing tokens to exit early at intermediate layers. However, their throughput is limited by the computational and memory savings. Existing EE-LLM frameworks rely on a single model and therefore, their token generation latencies are bottlenecked by tokens that do not exit early and traverse additional layers. Moreover, early exits are only known at runtime and depend on the request. Therefore, these frameworks load the weights of all model layers even though large portions remain unused when tokens exit early. The lack of memory savings limit us from scaling the batch sizes.

We propose *HELIOS*, a framework that improves both token generation latency and batch sizes to enable high-throughput in EE-LLMs. HELIOS exploits two insights. *First*, early exits are often complimentary across models, tokens that do not exit early on one model often take an early-exit on another. HELIOS employs multiple models and dynamically switches between them to collectively maximize the number of tokens that exit early, and minimize token generation latencies. *Second*, even when a predicted token does not exit early due to poor confidence, it often remains unchanged even after additional layer traversal. HELIOS greedily allows such tokens to exit early and only loads the weights of the most likely to be used layers, yielding memory savings which is then re-purposed to increase batch sizes. HELIOS employs real-time profiling to accurately identify the early-exit distributions, and adaptively switches between models by tracking tokens in real-time to minimize the performance degradation caused by greedy model loading and exiting. Our evaluations show that HELIOS achieves $1.48\times$ higher throughput and $15.14\times$ larger batch size compared to existing EE-LLM frameworks.

1 INTRODUCTION

Large Language Models (LLMs) presents critical *throughput* concerns, particularly as we adopt larger and complex models to produce more accurate and nuanced responses. Inference throughput can be increased by reducing token generation latencies and increasing batch sizes or number of concurrent requests served. *Early-Exit LLMs (EE-LLMs)* are a class of LLMs that allow tokens to exit at specific intermediate layers if their probability meets a confidence threshold. By skipping layers for simple tokens, EE-LLMs reduce the latency of token generation and improves throughput.

Limitations of Current EE-LLM Serving: Existing EE-LLMs rely on a single model and tokens that do not meet the confidence threshold must traverse additional layers. This limits latency savings for tokens that cannot exit early. Moreover, EE-LLMs cannot increase batch sizes due to two reasons. *First*, they do not yield memory savings because early exits taken depend on the request and are unknown apriori to serving it; and not all tokens exit early. Therefore, EE-LLMs load the weights of all layers on the GPUs and compute their Key-Value (KV) vectors to accommodate the worst-case exit. Figure 1 shows that the memory required to store the weights of two Llama models accounts for up to 68% of available HBM even on state-of-the-art NVIDIA B100s and is identical for vanilla auto-regressive decoding and EE-LLMs. Consequently, batch sizes cannot be increased because memory utilization remains unchanged.

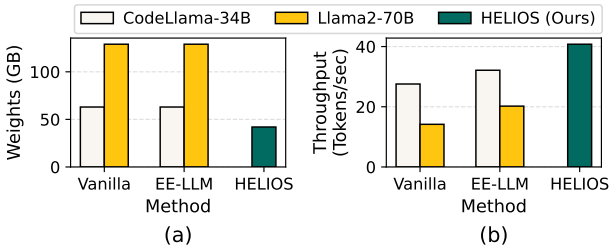

Figure 1. (a) Memory required to store model weights and (b) throughput of CodeLlama-34B and Llama2-70B models for vanilla auto-regressive decoding, EE-LLM, and HELIOS on ShareGPT (sha, 2023). By using multiple LLMs to maximize early exits across and greedily loading weights of most likely to be used layers, HELIOS reduces both token generation latencies and memory footprint. The memory savings lead to higher batch sizes and overall, HELIOS improves throughput by 45%, unlike EE-LLMs that only improve it by 16% relative to vanilla decoding.

Second, batched inference in EE-LLMs is non-trivial due to synchronization issues. Typically, in batching, a token is generated for every request, before proceeding to generate the next token for all requests in the batch. EE-LLMs must therefore, either wait for the token that takes the longest

[1]Department of Electrical and Computer Engineering, The University of Texas at Austin [2]NVIDIA. Correspondence to: Avinash Kumar <avinkumar@utexas.edu>, Shashank Nag <shashanknag@utexas.edu>.

in each round or simply use a batch size of 1. Given the synchronization overheads in the former, existing EE-LLMs use the latter by default (Chen et al., 2024; Pan et al., 2024).

Our Proposal: We propose *HELIOS*, a framework that improves both token generation latency and batch sizes to enable high-throughput in EE-LLMs. HELIOS, shown in Figure 2, exploits two key insights. *First*, tokens that do not exit early on one LLM often exits early on another. For example, our studies show that the first six layers of the 24-layer OPT-1.3B model processes 74% of tokens for a prompt mix of standard benchmarks, while the remaining 26% require all layers. However, 57% of these remaining tokens can be served by using only the first nine layers of the 32-layer OPT-6.7B model. Thus, HELIOS efficiently uses multiple LLMs to maximize the number of early exit tokens and lower the average token generation latencies drastically. For instance, by judiciously using both OPT-1.3B and 6.7B, HELIOS produces 92% of the tokens using early exits, compared to only 74% and 77% respectively by using them standalone. Note that significantly fewer tokens (only 8% in this case) now require additional layer traversal.

Second, our studies show that even if the confidence threshold is not met at an early exit, the predicted token often remains *unchanged* even after additional layer traversal. For example, our studies with CNN-Dailymail dataset (Nallapati et al., 2016) and OPT-6.7B model show that there is a 92.1% chance that tokens prevented from exiting at Layer-9 due to a low confidence score of 0.2, ultimately becomes the final output token after traversing through all 32 layers. Thus, most tokens that do not meet the confidence threshold can still be greedily obtained from early exits; and weights corresponding to the later layers would then remain largely unused. HELIOS leverages this insight to only load the weights of the most likely to be used layers (such as up to Layers 6 and 9 for the OPT-1.3B and 6.7B models respectively in the above example). This yields memory savings (3.37GB and 17.25GB respectively in this case) which is repurposed to support larger batch sizes. Note that this approach eliminates synchronization overheads between tokens in a batch because it ensures that each token only traverses a fixed number of model layers.

Key Challenges of HELIOS: Employing multiple LLMs, dynamically switching between them, and maximizing early exits is non-trivial due to several reasons. *First*, the early exits taken depend on the request and are unknown. *Second*, frequent model switching incurs overheads and causes slowdown. *Third*, aggressively allowing tokens to exit early when confidence is not met degrades accuracy. To address the first challenge, HELIOS employs real-time profiling to accurately identify the distribution of early exits for incoming requests. As successive requests often have overlapping contexts and exhibit locality (Lin et al., 2024), this early exit profile remains relevant and accurate temporally.

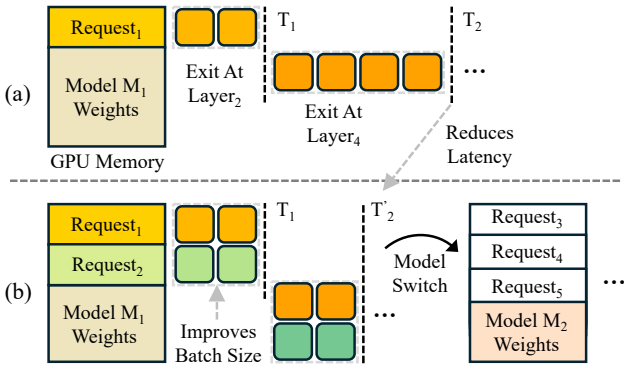

Figure 2. (a) Current EE-LLMs select a model (say M_1), load weights of all its layers, and use a batch size of 1 to avoid synchronization across tokens. (b) HELIOS uses multiple LLMs (M_1 and M_2 here) and only loads the weights of the layers most likely to be used based on real-time early exit profiles. HELIOS improves batch sizes by increasing available memory capacity and reducing synchronization overheads. HELIOS also monitors performance in real-time and switches between LLMs or loads additional layers of the current model to prevent accuracy degradation.

To minimize switching overheads and accuracy loss, HELIOS uses a two-step approach. *First*, it tracks a window of consecutive tokens and detects potential accuracy degradation if a certain number of tokens fail to meet the confidence threshold. *Second*, under such circumstances, HELIOS evaluates two options– either (1) load all layers of the model currently in use or (2) switch to an alternate model that can potentially complete the task more efficiently by using early exits. HELIOS evaluates the overheads of each option using the real-time telemetry data gathered during profiling and selects the option with minimal overheads.

Overall, this paper makes the following contributions:

1. We show that the throughput of EE-LLMs is limited because relying on a single LLM restricts computational savings for tokens that do not exit early, whereas the lack of memory savings limit us from scaling batch sizes.

2. We propose *HELIOS*, a framework that uses multiple models and dynamically switches between them to collectively maximize the number of early-exit tokens. This reduces token generation latencies and increases throughput.

3. HELIOS exploits the observation that low-confidence tokens which could not exit early, often remain unchanged even after additional layer traversal and steers more tokens to exit early, increasing throughput even further.

4. HELIOS profiles the early exit distributions in real-time and greedily loads the weights of only the most likely to be used layers. The memory savings enable higher batch sizes.

Our evaluations show that HELIOS improves throughput and batch sizes by $1.48\times$ and $15.14\times$ respectively compared to current EE-LLMs, with negligible impact on accuracy.

2 Background and Motivation

2.1 Early Exit Large Language Models

Early-Exit LLMs or *EE-LLMs* are a class of language models that enable high throughput inference by allowing tokens to exit at specific intermediate layers during the forward pass if their probability meets a predefined confidence threshold. EE-LLMs reduce the average token generation latency by skipping computations in the later model layers for simple tokens and improve throughput without degrading accuracy. The computational and latency savings with EE-LLMs scale proportionally with the number of layers skipped which eventually translate into higher throughput. Consequently, EE-LLMs have been widely adopted in various deep learning architectures in both industry and academia (Elhoushi et al., 2024; Wang et al., 2024; Xu et al., 2023).

2.2 Limitations of Existing EE-LLM Serving

Existing EE-LLM frameworks yield limited throughput benefits due to two key limitations. *First*, they yield *limited latency savings* because they rely on a single model and tokens that fail to exit early on this model must traverse additional layers until the confidence threshold is met. Consequently, the average token generation latency and throughput are limited by the number of tokens that do not exit early.

Second, EE-LLMs offer *no memory savings* which limits batch sizes. As early exits taken are only known at runtime and cannot be predicted in advance, current EE-LLMs load weights of all the model layers on to the GPUs, even though the later layers remain unused when tokens exit early. Note that despite recent advances in quantization (Zhao et al., 2024) and compression (Zhu et al., 2024), model weights dominate the GPU memory footprint. For example, the Llama3.1 405B model consumes about 52% of the available HBM to store model weights even on a node with eight state-of-the-art NVIDIA B100 GPUs. Moreover, current EE-LLM frameworks also compute and cache the Key-Value (KV) vectors for all skipped layers to account for the worst-case exit depth. This is because any future token that does not exit early, must attend to all preceding tokens. Thus, overall, the memory footprint of EE-LLMs is identical to vanilla auto-regressive decoding, limiting batch sizes.

2.3 Goal: Maximize Early Exit Tokens and Batch Size

As token generation latency depends on the number of layers traversed, ideally, we want to maximize the number of tokens that exit early to maximize throughput. To improve throughput even further, we must scale to larger batch sizes by improving the memory-efficiency such that we reduce the footprint of unused memory and re-purpose the memory savings to support additional requests in parallel. This paper proposes *HELIOS* that achieves these goals.

3 Our Proposal: HELIOS

In this paper, we propose *HELIOS*, a framework that improves the throughput of EE-LLMs by maximizing the number of early exit tokens, thereby reducing the average token generation latency, and increasing batch sizes through efficient memory management. Next, we discuss the key insights of our design before describing the implementation.

3.1 Key Insights

HELIOS leverages two key insights that are described next.

Insight-1 → Early exits are often complimentary: Our experiments show that early exits taken depend on the EE-LLM and are often complimentary across models. Tokens that require additional layer traversal or all layers of a model (i.e. no early exits taken) can often be predicted accurately with another model using early exits. For example, Figure 3 shows that prompts needing more than twelve layers on the 24-layer OPT-1.3B model require fewer layers on OPT-6.7B for similar accuracy. We make similar observations for prompts needing more than nine layers on the 32-layer OPT-6.7B model. HELIOS leverages this characteristic to employ multiple models and dynamically switches between them such that both the models collectively maximize the number of early exit tokens, thereby reducing the average token generation latency and improving throughput.

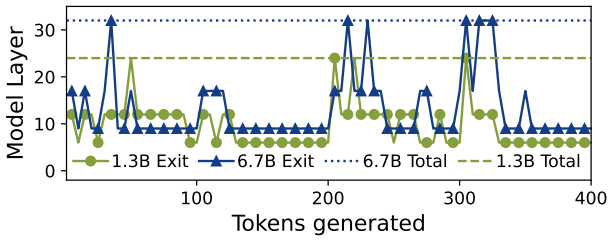

Figure 3. Exit layers for serving a typical workload with a mixture of prompts using OPT 1.3B and 6.7B models.

Insight-2 → Not meeting confidence is okay at times: Our studies reveal that even when the confidence threshold is not met at an early-exit, the predicted token frequently remains *unchanged*, even after traversing additional model layers. Figure 5(a) shows the fraction of tokens that remain unchanged even after traversing additional model layers of the 32-layer OPT 6.7B model as a function of their probability at the first early exit (Layer-9). For instance, if the confidence threshold is 1, no token would exit early. However, we observe that even if we consider the threshold to be as low as 0, the token predicted at Layer-9 is identical to the token predicted at the final exit Layer-32 for 85% of the cases; and traversing through subsequent layers only improves the confidence. A prior work (Zhou et al., 2020) also makes a similar observation.

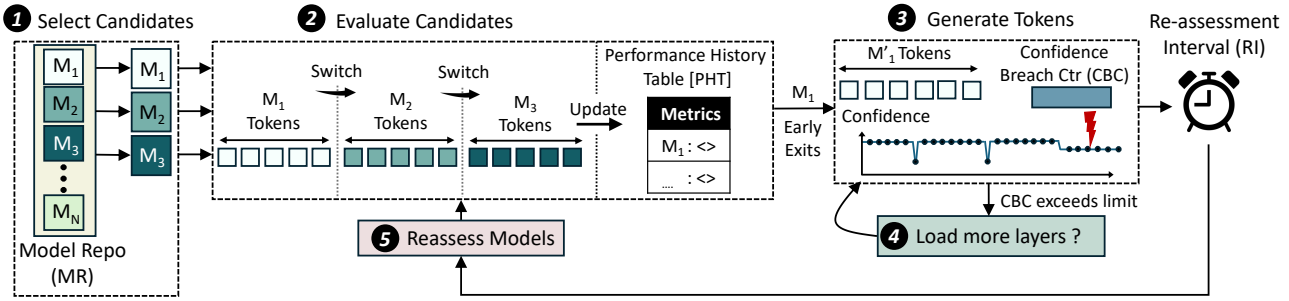

Figure 4. Design of HELIOS

Furthermore, early-exits consistently produce accurate output tokens even across different model sizes and families. For example, Figure 5(b) shows that even for Codellama-34B, 90% of the tokens produced by Layer-16 (one-third the model depth) remain unchanged for the CNN-Dailymail (Nallapati et al., 2016) dataset (more data provided in Appendix D). HELIOS leverages this insight to greedily load only the most likely to be used layers of the selected model and frequently allows tokens to exit even if the confidence threshold is not met, yielding memory savings that enable larger batch sizes. Note that the impact of this on accuracy is negligible because (1) HELIOS already drastically reduces the percentage of tokens requiring additional layer traversal by employing multiple models with complimentary early-exit characteristics and (2) not meeting the confidence threshold does not mean the predicted token is incorrect. HELIOS also introduces additional steps in the design to minimize accuracy loss, as will be discussed next.

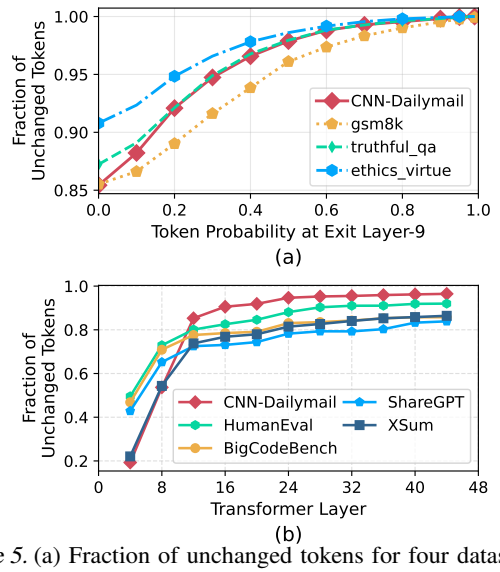

Figure 5. (a) Fraction of unchanged tokens for four datasets on OPT-6.7B model from the 1st exit layer (9) to the final layer (32). We observe that probability of the predicted token staying unchanged is always greater than 85%. (b) Fraction of tokens generated at an exit-layer that remain unchanged even after traversing the full-model for the Codellama-34B model across six datasets.

3.2 Design Overview

Figure 4 shows an overview of HELIOS. ❶ HELIOS selects a set of candidate models and ❷ evaluates their performance in real-time. ❸ The chosen model is then only loaded up to a limited number of layers based on the early exit history from the evaluation step and is used to generate tokens further. ❹ If the number of layers loaded for the current model are insufficient, the system requests for additional layers. HELIOS compares the overheads of (1) loading more layers for the current model versus (2) switching to another model from the candidate pool, and decides on one of them depending on their overheads. ❺ HELIOS also periodically reassesses the performance of the selected model and switches to another candidate model if needed, to adapt to the changing characteristics of the request stream.

3.3 Design Implementation

Next, we discuss the implementation of HELIOS.

3.3.1 Step-1: Selection of Candidate Models

Typically, service providers maintain a *Model Repository (MR)* containing key performance metrics, such as throughput, accuracy, across different standard benchmarks. HELIOS uses this telemetry data to select the *TopK* candidate models based on user-specified SLOs and available hardware. We illustrate this in Figure 4 using models M_1, M_2, and M_3 chosen as candidates. By default, HELIOS selects up to three models to minimize the time spent on the evaluation step, which is described next, and quickly converge on the suitable candidate for the current request stream.

3.3.2 Step-2: Evaluation of Candidate Models

Next, HELIOS evaluates the performance of the selected candidate models in real-time to obtain even more accurate telemetry data and gather the early-exit distribution profiles because they are not present in model repositories by default. This profile remains effective because successive queries often share *overlapping contexts* and exhibit locality. By default, HELIOS evaluates each candidate model for five

requests. Figure 4 illustrates this, where models M_1, M_2, and M_3 generates output tokens for five requests. This does not impact performance (such as time taken to generate first token or TTFT) because the output tokens generated are not discarded given that the preceding candidate selection process is highly selective and ensures that only competent models are shortlisted. Note that candidate model evaluation may also be done in parallel, but we avoid it in our default design due to hardware limitations (not enough GPUs).

Methodology: We obtain throughput and early-exits distribution using profiling tools. Assessing accuracy is non-trivial because incoming requests lack ground truth for comparison. So, we use *perplexity* because it is a reference-free, efficient, and suitable metric for real-time inference (details in Appendix A). Table 1 shows the perplexity of two models before and after evaluation and highlights the efficacy of our approach. If we were to select a model based on the repository data, we would select the OPT-6.7B model. However, we select the OPT-1.3B model because post-evaluation data suggests it to be more effective for the current prompts.

Table 1. Perplexity comparison between pre-determined value from MR in selection phase and post evaluation phase.

Model	Pre-Evaluation	Post-Evaluation
OPT-1.3B	1.91	1.47 ✓
OPT-6.7B	1.68 ✓	1.49

Performance History Table: The request-specific performance data (throughput, accuracy) and the early exit profiles of each model are saved in a table, called the *Performance History Table (PHT)*. The PHT is used in the subsequent stages of HELIOS, as will be discussed next.

3.3.3 Step-3: Token Generation Using Best Model

The best candidate model identified in the evaluation stage is used to generate tokens for the incoming requests.

Greedy Loading Up to Selected Exit Layers: HELIOS greedily loads the weights of the most likely to be used layers based on the early exit profile of the model. This yields memory savings which is used to increase batch sizes ***(Insight-#2)***. For example, Figure 6(a) shows the exits taken for a prompt mix with the OPT-1.3B model, where 74% of the requests only require six layers of the model. We refer to these as *Low-Exit Tokens (LTs)*. HELIOS greedily loads only up to six layers in this scenario (denoted by M_1' in Figure 4). If most pending requests are LTs that do not require additional layers, this greedy approach yields significant memory savings, without compromising accuracy.

Load More Layers Or Another Candidate Model? Although partially loaded models offer significant memory savings and improve batch sizes, we encounter tokens that do not meet the confidence threshold. For example, in Figure 6(a), 26% of the requests use more than six layers. We refer to these as *High-Exit Tokens (HTs)*. Under these circumstances, HELIOS has two options– (1) either load the remaining layers of the current LLM (OPT-1.3B in this case) or (2) switch to another model which can service the request stream with fewer layers. The first option ensures the current model is present in its entirety and guarantees that the confidence threshold will be met. In contrast, the second option aims to identify a more efficient alternative and maximize the total number of early-exits (***Insight-#1***). Figure 6(b) shows the distribution of the exits taken by the HTs when another candidate model, OPT-6.7B, is used. We observe that 57% of the HTs can be serviced by using only nine layers of the OPT-6.7B model. Note that the exit history information is already available in the PHT from Step-2. Loading more layers of the current model is beneficial only when the overall resource usage remains lower than the second option where another candidate model is loaded up to a limited number of layers (OPT-6.7B with nine layers in the example). Once the overheads of both options are evaluated, the option with minimal overheads is selected and appropriate action is taken.

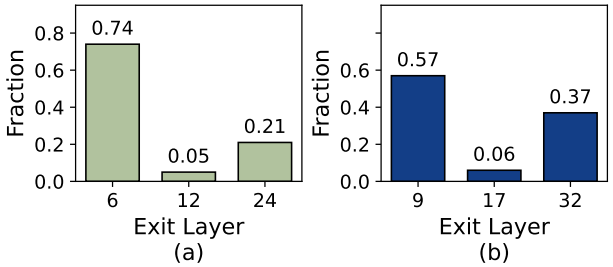

Figure 6. Distribution of exit layers for tokens of a prompt mix using the OPT-1.3B Model. 74% of the requests only require up to 6 layers. (b) Distribution of exit layers of the remaining 26% when they are serviced by OPT-6.7B model.

Amortizing Loading Overheads with CBC: Irrespective of the option selected, both approaches– loading additional layers and model switching, incur overheads. To minimize these overheads, HELIOS considers both options only after a certain number of tokens within a window fail to meet the confidence threshold (more details in Appendix E). This leverages our observation that not every token that fails to meet the confidence threshold at an early exit, actually changes after future layers (***Insight-#2***). HELIOS uses a *Confidence Breach Counter (CBC)* that increments whenever an output token does not meet the confidence threshold. If the CBC exceeds a pre-determined maximum allowable limit (CBC_{max}), HELIOS re-assesses whether it should load more layers or switch to another model to serve the incoming request stream. By default, HELIOS only tolerates up to 50 confidence threshold breaches ($\text{CBC}_{max} = 50$) in a window of 100 consecutive tokens.

3.3.4 *Step-4: Periodic Profiling to Maximize Early-Exits*

Ideally, we should re-evaluate early exit profiles for each request across all candidate models to maximize early-exits for the current request stream. However, frequent profiling introduces substantial overheads. In contrast, capturing the early-exit profile infrequently is not desirable because although input requests exhibit temporal locality, their characteristics often evolve over longer time periods (more details in Appendix B). Consequently, the early exit profile captured in the PHT that is currently being used by HELIOS may become obsolete and sub-optimal. HELIOS attains a sweet spot in this trade-off space by using the *Re-assessment Interval (RI)* hyperparameter and invokes the profiling phase (Step-2) periodically after RI requests have been served. By default, HELIOS collects early-exit profiles after every 150 requests (RI=150) but ideally, this hyper-parameter must be fine-tuned by the server provider to match the variations in their request streams. Our default implementation does not reselect candidate models because our evaluations show that some models generally perform better across a wide variety of tasks. For example, Figure 7 shows that Llama3-8B and Llama2-13B consistently perform well, whereas GPT2-124M is consistently poor. Nonetheless, HELIOS can also look up the model repository to select new candidate models, if the user-specified SLOs or hardware constraints change.

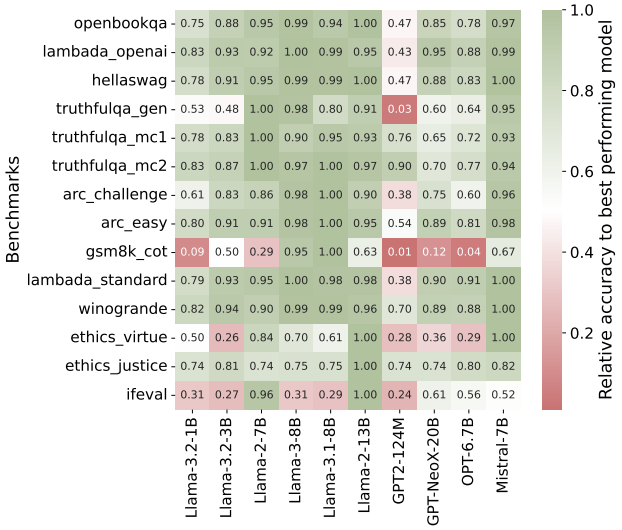

Figure 7. Accuracy of datasets across multiple models relative to the best model. We observe that some models consistently perform well (such as Llama3-8B, Llama2-13B) while some others (such as GPT2-124M) remain consistently poor. HELIOS exploits this observation to eliminate frequent candidate model selection and only performs this step when SLOs or hardware constraints change.

3.4 HELIOS Algorithm

Algorithm 1 describes the algorithm of HELIOS.

Algorithm 1 Adaptive EE-LLM Serving with HELIOS

Input: Model Repository(MR), SLO
Output: Dynamic Model Selection
Parameters: $\mathbf{M}$: Full Model; $\mathbf{M'}$: Low Exit Model; $\mathbf{CBC}$: Confidence Breach Counter; $\mathbf{CBC}_{\max}$: Threshold; $\mathbf{RI}$: Reassessment Interval;
Step-1: Candidates$\leftarrow Top_k$(MR(SLO, HW constraints))
while prompts in requests **do**
  $\mathbf{CBC}$, ServicedPrompts $\leftarrow 0$
  Step-2: PHT$[M, M'] \leftarrow$ Evaluate(Candidates)
  Chosen $\leftarrow$ BestModel(PHT$[M']$)
  repeat
    Step-3: Serve(prompt, Chosen)
    if confidence not met **then**
      CBC $\leftarrow$ CBC + 1
      if CBC $>$ CBC$_{\max}$ **then**
        if PHT$[M$(Chosen)$]$ $<$ PHT$[M'$(Others)$]$ **then**
          Chosen $\leftarrow M$[Chosen]
        else
          Chosen $\leftarrow M'$[NextBestModel(PHT)]
        end if
        CBC $\leftarrow 0$
      end if
    end if
  until ServicedPrompts $<$ RI (Step-4)
end while

4 Evaluation Methodology

We discuss the methodology used to evaluate HELIOS.

4.1 Models

We use publicly available early-exit variants of the Llama models hosted on HuggingFace (Elhoushi et al., 2024). Additionally, we augment and fine-tune two off-the-shelf models from Facebook's OPT family (Zhang et al., 2022), namely OPT-1.3B and OPT-6.7B, by incorporating early-exits at one-fourth the model depth. This strategy is consistent with prior works (Chen et al., 2024). We provide more details on models and fine-tuning in Appendix F. Table 2 summarizes the candidate model configurations used for our evaluations. By choosing diverse model families and sizes, our evaluations ensure that we capture diverse scenarios and shows the generalizabilty of our approach.

Table 2. Overview of Candidate Model Configurations

Configuration	Candidate Models
1	OPT-1.3B & OPT-6.7B
2	Llama2-7B, 13B & Llama3-8B
3	CodeLlama-34B & Llama2-70B

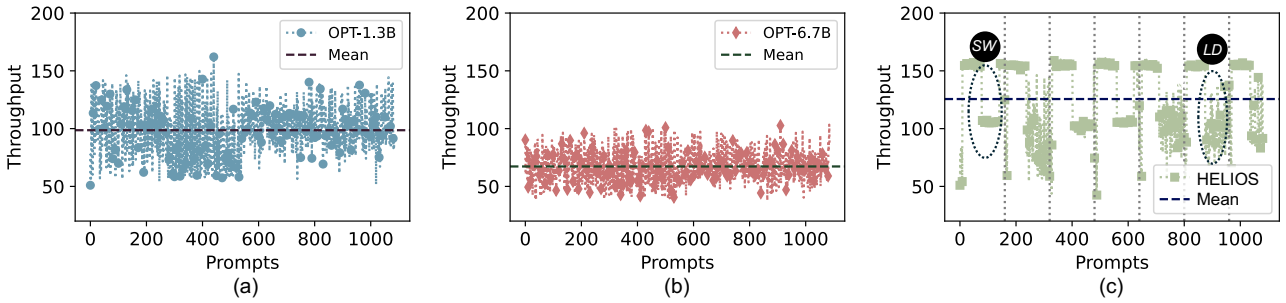

Figure 8. Comparison of *Throughput* when using (a) only OPT-1.3B (EE-LLM), (b) only OPT-6.7B (EE-LLM), and (c) HELIOS (*higher is better*). Candidate re-assessment steps are denoted using vertical dotted lines in (c).

4.2 Setup

We use the EE-LLM framework (Chen et al., 2024) for both fine-tuning and inference serving, which is consistent with prior works (Chen et al., 2024; Pan et al., 2024). We limit our evaluations to three models due to limited access to GPUs. We use a node equipped with four NVIDIA A100 (40 GB) GPUs and 64-core AMD EPYC CPU. The GPUs are inter-connected via an all-to-all NVLink, providing upto 400 GB/s bidirectional bandwidth. Due to the large model weights of Llama2-70B and CodeLlama-34B, tensor parallelism of 4 and 2 is applied for these models, respectively, whereas all other models use a tensor parallelism of unity.

4.3 Datasets

To evaluate the efficacy of HELIOS at a server scale, we use the ShareGPT (sha, 2023) dataset which comprises multi-turn user-LLM interactions. ShareGPT is a representative workload for analyzing system performance at server-scale as it is directly sourced from a live inference server. We further evaluate HELIOS on a request stream composed of a wide range of tasks. This mixture draws requests from standard benchmarks including CNN Daily Mail (Nallapati et al., 2016), gsm8k(Cobbe et al., 2021), CodeXGLUE (Lu et al., 2021) and HellaSwag (Zellers et al., 2019). We provide additional details about these datasets in Appendix C.

4.4 Figure-of-Merit

We use perplexity to evaluate the accuracy of the generated tokens. This is consistent with prior works (Federici et al., 2024; Frantar & Alistarh, 2023; Xu et al., 2024; Ma et al., 2023) (additional details in Appendix A). To assess server performance, we use other widely used metrics such as Time Taken to First Token (TTFT), Time Per Output Token (TPOT), latency, and throughput, as summarized in Table 3. Our primary evaluation of HELIOS assumes *improving throughput* as the user-specified SLO. But we also consider other SLOs and provide experimental results in Appendix G to show that HELIOS is generalizable.

Table 3. Summary of Metrics Used

Metric	Specification
Perplexity	Captures coherence in output tokens
TTFT	Time Taken to First Token
TPOT	Time Taken Per Output Token
Latency	TTFT + TPOT $\times$ Number of Tokens
Throughput	Tokens generated per second ($\frac{1}{TPOT}$)
Batch Size	Number of requests served in parallel

5 Results

By default, we consider the user's SLO is to maximize the inference throughput.

5.1 Throughput

For this particular evaluation, we restrict HELIOS to a batch size of 1 because it allows us to thoroughly evaluate the throughput benefits that solely stem from maximizing early exits. We also consider a prompt mix of standard benchmarks to evaluate the generalizability. The throughput is inversely proportional to the number of layers traversed and time spent per layer. Thus, it increases if more tokens (1) take early exits and (2) we use smaller models for early exits because traversing a layer takes longer on larger models. Figure 8 shows the *throughput (higher is better)*. HELIOS improves the throughput by $1.48\times$ and $2.13\times$ on average compared to using the OPT-1.3B and OPT-6.7B models standalone EE-LLMs respectively. In Figure 8(c), *LD* illustrates scenarios where more layers of the current model are loaded, whereas *SW* denotes cases where HELIOS switches to another model, highlighting the efficacy of the adaptive nature of HELIOS.

Table 4 compares the exit distribution of tokens when the OPT EE-LLMs are used standalone against HELIOS. In HELIOS, about 91% of the tokens are processed using the earliest exits of both models combined (Layer-6 of the OPT-1.3B model and Layer-9 of the OPT-6.7B model), compared

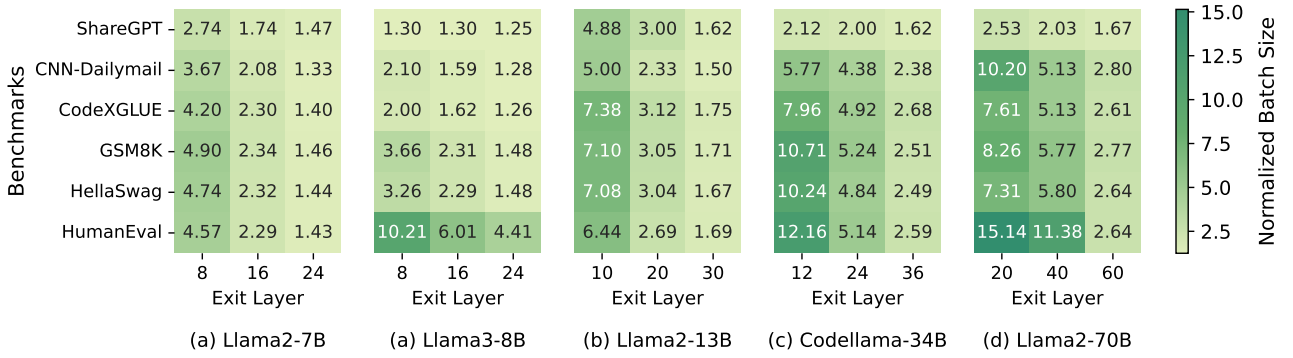

Figure 9. Normalized batch size with HELIOS compared to current EE-LLM framework. Greedily loading only most-likely to be used layers reduces the footprint of both the weights and Key-Value (KV) caches, yielding significant memory savings which are repurposed to support other requests in parallel and increasing the batch sizes.

to only about 73% while using the models standalone. Also, a significant portion (about 70%) of these tokens are processed using the earliest exit of the smaller OPT-1.3B model. The percentage of requests that use all layers of both models combined is only 7.39%, 3× lower than using either EE-LLM standalone. This highlights the efficacy of HELIOS in maximizing the number of early exit tokens and ensuring that the accuracy impact is minimal because tokens that require additional layers are not compromised. Furthermore, HELIOS also has negligible impact on accuracy compared to existing EE-LLMs. Our experiments show that the achieved perplexity in HELIOS for the prompt mix is only 0.01 higher than the OPT-1.3B model. Note that for this particular case the OPT-1.3B model is more accurate than the OPT-6.7B model and is thus used for comparison.

Table 4. Comparison of the percentage of tokens processed by different exit layers for different model selection methods.

Model Selection	OPT-1.3B			OPT-6.7B		
	6	12	24	9	17	32
OPT-1.3B Only	73.0	4.70	22.3	-	-	-
OPT-6.7B Only	-	-	-	73.6	4.80	21.6
HELIOS	70.19	1.38	6.78	20.90	0.14	0.61

5.2 Response Time and Latency

Table 5 compares the Time To First Token (TTFT) and Time Per Ouput Token (TPOT) for HELIOS against each EE-LLM standalone. We observe that the TTFT of HELIOS is significantly lower because by default it serves most requests using a smaller model with early exits. Similarly, the token generation latency or TPOT is up to 46.6% lower because HELIOS maximizes the total number of early-exit tokens.

Table 5. Comparison of response time and latency for the standalone EE-LLM models and HELIOS.

Model	TTFT (s)	TPOT (s)
OPT-1.3B Only	0.042	0.010
OPT-6.7B Only	0.069	0.015
HELIOS	0.033	0.008

5.3 Batch Sizes

HELIOS supports larger batch sizes by (1) eliminating synchronization overheads and (2) re-purposing the GPU memory saved via greedy loading of weights of only the most likely to be used layers. In HELIOS, all tokens at any given timestep must exit the same early exit layer (up to which the selected model is loaded), thus completely eliminating the need for synchronization. Processing a request requires memory to store (1) model weights, and (2) key-value (KV) caches for each layer of the model to generate tokens. While model weights can be shared, each request must maintain its own KV-caches. Loading only a subset of layers yields substantial memory savings, as the memory footprint of both the model weights and KV cache decreases in proportion to the number of layers skipped from loading. This yields considerable memory savings, which is repurposed by HELIOS to allocate KV cache space for additional requests. This enables HELIOS to support larger batch sizes.

Figure 9 compares the batch sizes with HELIOS against current EE-LLM frameworks. Here, we consider additional benchmarks beyond the prompt mix considered earlier. HELIOS improves batch sizes by up to 15.14×, highlighting its efficacy. Table 6 compares the memory footprint of HELIOS against each EE-LLM standalone. HELIOS yields significant memory savings (up to 67.4%). Note that for the scenario where CodeLlama-34B and Llama2-70B are used,

HELIOS is not required to load the entire model at all.

Typically, HELIOS yields greater memory savings for larger models comprising more early exits. This is because weights occupy a substantial amount of GPU memory for large models. For example, the Llama2-70B model occupies 81.6% of the available memory in our setup with 160 GB. Moreover, larger models offer greater flexibility for early exits. For example, fine-tuning early-exits uniformly gives nine early exit paths in the Llama2-70B model, compared to only three in the Llama2-7B model.

Table 6. Memory footprint on Nvidia A100 GPUs for individual models standalone and HELIOS for the ShareGPT dataset.

Model	Weights Memory Size (GB)
Using only two candidate models	
CodeLlama-34B Only	63
Llama2-70B Only	129
HELIOS	*42*
Using three candidate models	
Llama2-7B Only	12.9
Llama3-8B Only	15.5
Llama2-13B Only	24.8
HELIOS	*11.5*

Note that the memory savings in HELIOS are orthogonal to other memory optimization methods that reduce the size of the weights memory through quantization (Zhao et al., 2024; Liu et al., 2024) and KV caches (Li et al., 2024; Ghadia et al., 2025; Zhang et al., 2023; Xiao et al., 2023) independently. HELIOS can be combined with these approaches for even greater memory savings and higher throughput benefits.

5.4 Accuracy of Down-Stream Tasks

HELIOS does not degrade accuracy of downstream tasks. This is because the accuracy of downstream tasks typically saturates after traversing a specific model depth. For example, Figure 10 shows that the accuracy of the CodeLlama-34B model (with total 48 layers) saturates at layer-28 across several benchmarks. Similarly, accuracy of the Llama2-7B model (with total 32 layers) plateau's at layer-24. In fact, prior work (Gromov et al., 2024) shows that *nearly half* of the model layers can be skipped without affecting the downstream accuracy of most tasks for the Llama2-70B model. While HELIOS greedily loads model layers, it *ensures that it loads weights corresponding to layers which are most likely to be used* based on real-time early-exit profiles. This allows HELIOS to improve throughput while preserving downstream task accuracy despite its greedy nature.

To show this, we evaluate HELIOS in a configuration with three candidate models (Llama2-7B, Llama3-8B, Llama2-13B). Table 7 shows that the accuracy of HELIOS remains comparable, if not identical, across all benchmarks.

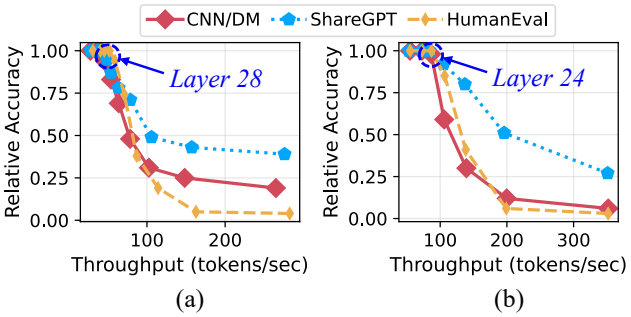

Figure 10. Relative accuracy versus throughput (tokens/sec) for (a) CodeLlama-34B, and (b) Llama2-7B across evaluation benchmarks for varying layer depths. Accuracy plateaus at layer 28 for CodeLlama-34B and layer 24 for Llama2-7B.

Table 7. Comparison of downstream accuracy (ROUGE-2 score) between HELIOS and full-depth baselines.

Model	ShareGPT	CNN/DM	HumanEval
Llama2-7B	0.097	0.077	0.041
Llama3-8B	0.098	*0.099*	0.054
Llama2-13B	0.098	0.089	0.058
HELIOS	*0.098*	0.097	*0.058*

5.5 Scalability Across Confidence Thresholds

The predefined confidence threshold (TH) for early exiting is a critical parameter that dictates the performance of EE-LLMs and HELIOS. Increasing the confidence threshold reduces the number of tokens taking early exits in existing EE-LLM serving frameworks because it becomes much harder to meet the exit criterion. Consequently, more tokens traverse additional layers, eventually reducing the effective throughput. This is consistent with our observation in Figure 11 which shows the impact of increasing confidence thresholds on the throughput for two different EE-LLMs (OPT-1.3B and OPT-6.7B).

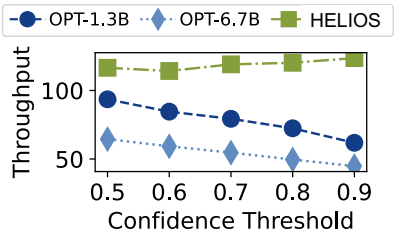

Figure 11. Impact of increasing TH on throughput.

In contrast, the throughput benefits of HELIOS remains consistent. This is due to two reasons. *First*, by employing multiple models, HELIOS encounters more tokens that naturally meet the confidence threshold and exits. *Second*, by greedily loading weights of only those layers that are most likely to be used and enforcing tokens to breach the confidence thresholds, the impact of increasing confidence thresholds on throughput is minimized in HELIOS.

5.6 Impact of Re-assessment

The Re-assessment Interval (RI) hyper-parameter impacts the performance of HELIOS– too low values of RI triggers frequent re-evaluation of the early-exit profiles and incur overheads, whereas large values imply that HELIOS may be potentially serving requests with obsolete exit profiles. Figure 12 shows the throughput for increasing values of RI. Existing EE-LLM frameworks do not have any notion of RI and thus, their throughput remains unimpacted. For HELIOS, the throughput is highest for RI=50 and reduces up to 250. Also, higher values of RI yields higher throughput but there is also a high likelihood that this impacts accuracy, because the nature of the incoming prompts may change between two re-assessment phases. These may go undetected if HELIOS never exceeds the Confidence Breach Counter during this timeframe. Our default implementation uses an RI of 150 because HELIOS also involves additional model switching due to the Confidence Breach Counter exceeding its maximum tolerable limit in between re-assessment intervals. Thus, by picking an RI slightly above 50 allows us to minimize the overheads of re-assessment without largely impacting throughput and accuracy.

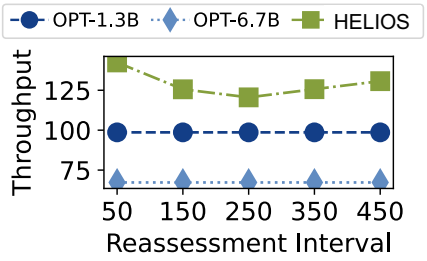

Figure 12. Impact of increasing RI on throughput.

5.7 Impact of Loading Layers and Model Switching

HELIOS greedily loads most likely to be used model layers and allows tokens to exit-early even when the confidence threshold is not met. However, to maintain accuracy, it tracks confidence violations through the Confidence Breach Counter (*CBC*). If the CBC detects a series of confidence violations within a given time period, HELIOS must decide on whether to (1) load additional layers of the current model, or (2) switch to an alternate candidate model at runtime. These mechanisms allow HELIOS to preserve accuracy without sacrificing the throughput benefits from greedily taking early-exits. To isolate the contribution of these mechanisms, we evaluate the throughput and accuracy (perplexity) of HELIOS under three configurations:

1. **HELIOS without loading**: HELIOS switches to another model if confidence threshold is not met.

2. **HELIOS without switching:** Loads additional layers of the current model if confidence is breached.

3. **Default implementation**: Allows both loading additional layers and model switching.

For this study, we use the OPT-1.3B and OPT-6.7B models. Table 8 shows that the while configuration-I (without loading) achieves the highest throughput, it also has the highest perplexity. On the other hand, configuration-II (without switching) has lower perplexity, but reduces performance as it cannot switch to the optimal model configuration at runtime to maximize early-exits. The default implementation of HELIOS strikes a balance by dynamically selecting between both options depending on the varying characteristics of the request stream.

Table 8. Comparison of throughput (higher is better) and perplexity (lower is better) for standalone EE-LLM baselines and three configurations of HELIOS on the ShareGPT dataset.

Configuration	Throughput	Accuracy
OPT-1.3B Only	98.5	1.61
OPT-6.7B Only	67.2	1.58
HELIOS (w/o Loading)	142.9	1.65
HELIOS (w/o Switching)	135.2	1.63
HELIOS (Default)	*138.5*	*1.62*

5.8 Safeguards Against Non-Deterministic Spikes

HELIOS dynamically loads layers and switches models at runtime based on the evolving pattern of the request stream. This can introduce non-deterministic latency spikes due to runtime memory allocation, data transfer stalls, and memory fragmentation. These spikes can directly violate latency SLOs. To overcome these non-deterministic latency spikes, HELIOS employs three key optimizations:

1. **Pre-allocated Memory Pool:** HELIOS is implemented using the pytorch library and leverages the native memory allocator to maintain a pool of pre-allocated GPU HBM. By executing transitions (whether loading more layers or switching to a different model) within this pre-allocated space, HELIOS avoids runtime cudaMalloc calls, thereby eliminating latency spikes associated with page table updates, TLB misses, and memory fragmentation.

2. **Overlapping compute and memory access during loading:** HELIOS overlaps the transfer of newly requested (additional) layers with the execution of the currently resident layers. This approach enables HELIOS to hide the latency overheads of loading additional layers. For example, we observe that loading 4 additional layers for the Llama3-8B model when 12 layers are already resident in GPU HBM only incurs a mere overhead of 4ms.

3. **Greedy loading to amortize switching costs:** While switching to another model, HELIOS only loads the most likely to be used layers in the new model based on the early-exit distribution from its profiling phase. Moreover, HELIOS pipelines this step by overlapping the loading of subsequent layers with the execution of layers that are already in memory to amortize model switching overheads.

5.9 Generalization to Other SLOs

The results presented in this section so far considers the user default SLO is to maximize throughput. However, HELIOS is generalizable and can effectively accommodate a wide-range of SLO objectives. To evaluate this, we also consider other SLOs, such as response time, accuracy, and energy-efficiency. We provide a comprehensive analysis of our results in the Appendix G.

6 Related Work

Prior software and hardware works explore trade-offs across LLM performance metrics, we compare and contrast below:

Comparison with Speculative Decoding: Speculative Decoding (Leviathan et al., 2023; Kim et al., 2023; Stern et al., 2018) is an inference-time technique which orchestrates two models, a smaller draft model, and a larger target model. The draft model generates output tokens one-by-one, while the target model periodically verifies and corrects them in parallel. However, unlike speculative decoding, early-exits significantly reduce energy-consumption by skipping further layers of the same model, and eliminating the need for a computationally intensive verification phase. Our experiments indicate that a speculative system consisting of OPT-125M, and OPT-6.7B consumes 1.49× more energy compared to OPT-6.7B with two early exits on the CNN Daily Mail (Nallapati et al., 2016) dataset. LayerSkip (Elhoushi et al., 2024) incorporates early-exits into their speculative decoding framework for further benefits. Furthermore, the selected draft layer in Layerskip remains static throughout execution, regardless of variations in the nature of inputs within the request stream. HELIOS adaptively switches between EE-LLM models or loads more layers to collectively maximize early-exits. Single exit-layer based speculative decoding can still be adopted by HELIOS if the served model being served contains atleast one intermediate exit prior to reaching the depth of the greedily loaded EE-LLM.

Hardware-Software Co-Design: BERT Loses Patience (Zhou et al., 2020) proposes a technique for fast and robust inference by introducing a patience-based early exit. In this approach, auxiliary classifiers are attached to intermediate layers, and the forward pass is dynamically terminated if the predicted class remains the same across consecutive layers. This approach leverages several layers to trigger the termination of a forward pass for low-confidence tokens. In contrast, HELIOS uses early-exit distributions in real-time to greedily only loads the most likely to be used layers in GPU memory. This allows HELIOS to not only improve the token-generation latency, but also support larger batch sizes due to the memory savings yielded by its greedy approach. Prior work, Edge-BERT (Tambe et al., 2021) proposes exploiting GPU power-management to manage the clock frequency based on predicted early exits. This allows Edge-BERT to save substantial power when tokens exit after traversing very few transformer layers. HELIOS is a generalized framework capable of accommodating any user-defined SLO, rather than being restricted to energy-efficiency, which is the primary focus of Edge-BERT.

Model Serving Optimizations: In addition, there have been several works on LLM inference serving at the cloud. INFaaS (Romero et al., 2021) selects a model that meets SLOs of the task and performs inference with it, and Clipper (Crankshaw et al., 2017) combines predictions from multiple models hosted concurrently. HELIOS instead dynamically selects a model to get predictions from a single model at a time, while ensuring the overall perplexity remains similar. Techniques like pipeline parallelism (Narayanan et al., 2019), and model parallelism, as used in AlpaServe (Li et al., 2023) are complementary to HELIOS, and could be combined to scale our design to accommodate larger models on the GPUs. Similarly Splitwise (Patel et al., 2024), a technique which splits the prefill and the token generation phase across multiple GPUs complements the design of HELIOS.

Dynamic Neural Networks: Multiple works have developed neural networks that have different computational graphs, based on particular deployment and prompt scenarios (Han et al., 2022; Veit & Belongie, 2017). However unlike HELIOS, these frameworks only consider a single model and the configurations within it.

7 Conclusion

Early-Exit LLMs (EE-LLMs) are promising variants of LLMs that enable high throughput inference by allowing tokens to exit at specific intermediate layers during the forward pass if their probability meets a predefined confidence threshold. This makes them attractive as they improve throughput without compromising accuracy. Existing EE-LLM frameworks rely on a single model and thus, their token generation latencies are primarily limited by the tokens that do not exit early. To accommodate the worst case exit-depth, current EE-LLM serving frameworks load the weights of all model layers, even though the memory corresponding to the later layers remain unused when tokens exit early. Limited latency savings and poor memory management severely limits us from attaining large throughput benefits in these frameworks.

In this paper, we propose HELIOS, an adaptive EE-LLM serving framework that yields reduced token generation latencies by maximizing the total number of early-exit tokens; and improves memory efficiency, enabling us to scale batch sizes. HELIOS orchestrates multiple models and dynamically switches between them to collectively maximize early-exits for a given set of input prompts. Furthermore, it exploits the insight that low-confidence tokens that do not take early exits often remain unchanged even after additional layer traversal. Therefor, HELIOS leverages greedily loads only the weights of the most likely to be used layers onto the GPU memories, yielding memory savings, which are then repurposed to support larger batch sizes. Our studies show that HELIOS achieves $1.48\times$ throughput and up to $15.14\times$ higher batch size compared to existing frameworks while meeting SLOs with negligible impact on accuracy.

ACKNOWLEDGMENTS

The authors acknowledge the Texas Advanced Computing Center (TACC) and the Center for Generative AI at the University of Texas at Austin for providing computational resources that helped develop the research results reported in this paper. We thank the generous support from the Cockrell School of Engineering and the Amazon AI PhD Fellowship Program through the Amazon Science Hub at UT Austin. This research was supported in part by NSF Grants #2326894 and #2425655, and the NVIDIA Applied Research Accelerator Program Grant. Poulami Das acknowledges the generous support through the AMD endowment at the University of Texas at Austin.

REFERENCES

Hugging face,, 2016. https://huggingface.co/.

Tensorflow serving for model deployment in production, 2018. https://www.tensorflow.org/tfx/guide/serving.

Sharegpt, 2023. https://sharegpt.com.

Aws: Prompt cache, 2024. https://aws.amazon.com/bedrock/prompt-caching/.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, Y., Pan, X., Li, Y., Ding, B., and Zhou, J. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. In *The Forty-first International Conference on Machine Learning*, 2024.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Computer, T. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A {Low-Latency} online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 613–627, 2017.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Elhoushi, M., Shrivastava, A., Liskovich, D., Hosmer, B., Wasti, B., Lai, L., Mahmoud, A., Acun, B., Agarwal, S., Roman, A., et al. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

Federici, M., Belli, D., Van Baalen, M., Jalalirad, A., Skliar, A., Major, B., Nagel, M., and Whatmough, P. Efficient llm inference using dynamic input pruning and cache-aware masking. *arXiv preprint arXiv:2412.01380*, 2024.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Ghadia, R., Kumar, A., Jain, G., Nair, P., and Das, P. Dialogue without limits: Constant-sized kv caches for extended responses in llms. *arXiv preprint arXiv:2503.00979*, 2025.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic Neural Networks: A Survey . *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 44(11):7436–7456, November 2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2021.3117837. URL https://doi.ieeecomputersociety.org/10.1109/TPAMI.2021.3117837.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., and Keutzer, K. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36:39236–39256, 2023.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., and Stoica, I. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving, 2023. URL https://arxiv.org/abs/2302.11665.

Lin, C., Han, Z., Zhang, C., Yang, Y., Yang, F., Chen, C., and Qiu, L. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945, 2024.

Liu, Z., Zhao, C., Fedorov, I., Soran, B., Choudhary, D., Krishnamoorthi, R., Chandra, V., Tian, Y., and Blankevoort, T. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*, 2024.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C. B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S., and Liu, S. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664, 2021.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Nallapati, R., Zhou, B., dos Santos, C., Gulcehre, C., and Xiang, B. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Riezler, S. and Goldberg, Y. (eds.), *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pp. 1–15, 2019.

Pan, X., Chen, Y., Li, Y., Ding, B., and Zhou, J. Ee-tuning: An economical yet scalable solution for tuning early-exit large language models. *arXiv preprint arXiv:2402.00518*, 2024.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Romero, F., Li, Q., Yadwadkar, N. J., and Kozyrakis, C. {INFaaS}: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 397–411, 2021.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Stern, M., Shazeer, N., and Uszkoreit, J. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P. N., Rush, A. M., Brooks, D., and Wei, G.-Y. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*. Association for Computing Machinery, 2021.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P.,

Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Veit, A. and Belongie, S. J. Convolutional networks with adaptive computation graphs. *CoRR*, abs/1711.11503, 2017. URL http://arxiv.org/abs/1711.11503.

Wang, J., Li, B., and Zhang, G. L. Early-exit with class exclusion for efficient inference of neural networks. In *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pp. 263–267. IEEE, 2024.

Wang, Y., Pan, Y., Yan, M., Su, Z., and Luan, T. H. A survey on chatgpt: Ai-generated contents, challenges, and solutions. *IEEE Open Journal of the Computer Society*, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Xin, J., Tang, R., Yu, Y., and Lin, J. Berxit: Early exiting for bert with better fine-tuning and extension to regression. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, pp. 91–104, 2021.

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

Xu, P., Shao, W., Chen, M., Tang, S., Zhang, K., Gao, P., An, F., Qiao, Y., and Luo, P. Besa: Pruning large language models with blockwise parameter-efficient sparsity allocation. *arXiv preprint arXiv:2402.16880*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. Opt: Open pre-trained transformer language models, 2022.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Zhao, Y., Lin, C.-Y., Zhu, K., Ye, Z., Chen, L., Zheng, S., Ceze, L., Krishnamurthy, A., Chen, T., and Kasikci, B. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Li, T., Zhuang, S., Wu, Z., Zhuang, Y., Li, Z., Lin, Z., Xing, E. P., et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. *arXiv preprint arXiv:2309.11998*, 2023.

Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., and Wei, F. Bert loses patience: Fast and robust inference with early exit, 2020. URL https://arxiv.org/abs/2006.04152.

Zhu, X., Li, J., Liu, Y., Ma, C., and Wang, W. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024.

A USING PERPLEXITY FOR ACCURACY

HELIOS performs inference-time accuracy estimation to decide when to selectively load more layers or switch to a different model. Accuracy estimation methods can be grouped into three-main categories:

- **Reference-Based Metrics:** This category of metrics rely on pre-defined ground truth labels to evaluate the quality of generated text. They measure the degree of overlap between the generated and the ground-truth reference using lexical or semantic similarity measures such as ROUGE, BLEU, and METEOR.
- **Reference-Free Metrics:** These metrics evaluate the quality of generated text without relying on reference outputs. Instead, they asses aspects such as fluency and coherence directly from the model output itself, using measures like Perplexity, BLANC, and Supert.
- **LLM-Based Metrics:** These metrics leverage LLMs for evaluating the quality of generated text. LLM-based evaluators are typically prompt-driven and can operate in both reference-free and reference-based scenarios. Common frameworks include Reason-then-Score (RTS), Head-to-Head (H2H), and G-Eval.

As HELIOS is an inference-time technique and ground-truths for incoming requests are unavailable to the server, it is limited to employing reference-free methods for accuracy estimation. While LLM based-metrics provide with detailed assessments of output quality, their usage is impractical because they require loading a separate, much larger LLM in memory than the one being evaluated.

We use *Perplexity* to estimate accuracy in real-time. Unlike metrics such as BLANC which measure how the output text aids the language model to reconstruct the source document, Perplexity captures a models own confidence over its output distribution. BLANC involves repeated masking and re-encoding operations, making it unsuitable for real-time inference. In contrast, Perplexity is computed directly from token probabilities, providing an intrinsic, reference-free measure of output quality that can be efficiently computed during inference. Furthermore, our evaluations ensure that perplexity comparison between models is fair by restricting the model repository to only those models which have the same tokenizer and vocabulary, thereby maintaining consistency in tokenization and likelihood normalization.

B LOCALITY IN THE REQUEST STREAM

Traffic in the request stream is frequently dominated by multi-turn sessions (Zheng et al., 2023) where users typically submit several queries with incremental adjustments to steer the LLM toward providing more nuanced responses (Madaan et al., 2023). In fact, this property is used by *several* state-of-the-art inference engines (AWS, 2024; ten, 2018) to improve caching mechanisms. Besides this natural behavior, existing prompt engineering techniques such as few-shot prompting, chain-of-though, or the inclusion of static system instructions serve as a second, independent source of locality (Wei et al., 2022; Brown et al., 2020). Consequently, the early-exit distributions captured during the profiling phase remain relevant and temporally accurate. However, HELIOS also periodically profiles candidate models and early-exit distributions to adapt to the changing characteristics of the request stream over a longer time-span.

Although, high-frequency task interleaving in the request stream, where successive requests frequently contain orthogonal queries could disrupt temporal locality, HELIOS outperforms the baseline even in such scenarios. This is because, baseline EE-LLM serving incurs the worst-case latency by traversing all model layers for each token regardless of prediction complexity. Whereas, HELIOS dynamically remaps segments of the request stream to the most suited candidate model. Any overhead introduced due to model switching in HELIOS is significantly lower than the cumulative latency incurred by the baseline due to additional traversal for simpler tokens.

C BENCHMARKS USED FOR EVALUATION

We evaluate HELIOS across a wide-range of LLM tasks including conversation, summarization, mathematical reasoning, code-generation, and sentence completion. The following section provides detailed descriptions of the benchmarks employed in our evaluations:

Conversation: ShareGPT (sha, 2023) is a collection of publicly available user-shared conversation logs between humans and a LLM. These logs have been used by various open-source initiatives for training and evaluating LLMs.

Summarization: The CNN/Dailymail dataset (Nallapati et al., 2016) is a large-scale summarization benchmark. It comprises news articles sourced from the CNN and the Dailymail website.

Code-Generation: The CodeXGLUE dataset (Lu et al., 2021) is a comprehensive benchmark encompassing a variety of coding tasks including completion and translation across several programming languages.

Mathematical Understanding: To asses arithmetic and reasoning capabilities, we employ the GSM8K dataset (Cobbe et al., 2021). It consists of 8.5K grade school level math word problems. Each problem requires multi-step reasoning to arrive at the numerical answer, testing the LLMs ability to deduce logic and perform basic mathematical computation.

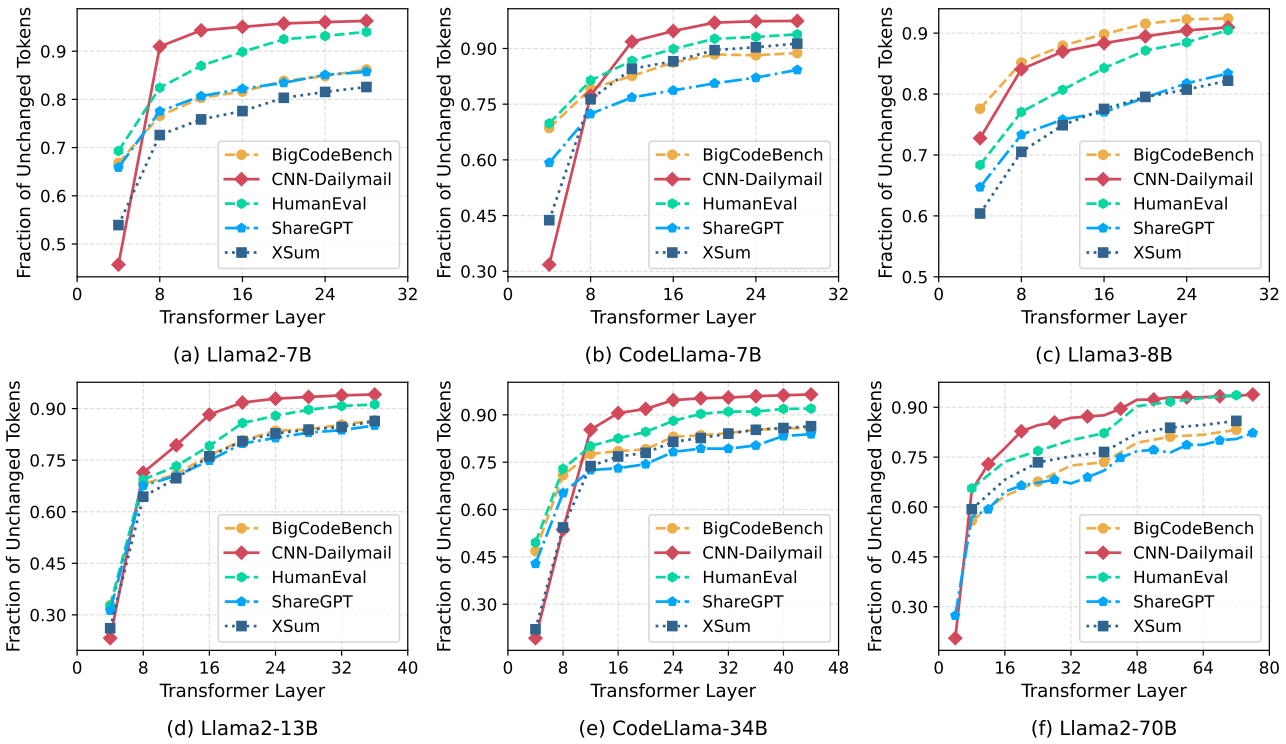

Figure 13. Fraction of unchanged tokens produced by an exit layer that remain unchanged even after additional model traversal. We observe that the probability that a predicted token remains unchanged reaches up to 90% for the Llama2-7B model as early as layer 8 i.e., one fourth the model depth. Furthermore, this trend remains consistent across models of various sizes from the Llama family.

Sentence Completion: The HellaSwag (Zellers et al., 2019) dataset is designed to test the sentence completion ability of a LLM. Each example provides a short-narrative followed by many continuations, only one of which forms a coherent and context-appropriate completion.

Figure 14 compares the token-level entropy across these datasets. Higher entropy indicates lower predictability and greater lexical diversity, while lower entropy corresponds to higher predictability and a more repetitive structure. These datasets exhibit consistently high entropy, suggesting that accurately predicting output tokens is non-trivial. Overall, the usage of these datasets ensures that our evaluation maintains a consistent level of predictive challenge.

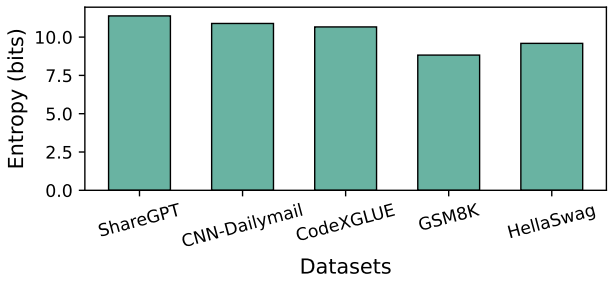

Figure 14. Comparison of dataset entropy. Higher entropy indicates lower predictability and greater diversity in lexical structure.

D Low-Confidence Tokens Remain Unchanged

Transformer-based LLMs are composed of several decoder layers. Each layer processes the output of the previous layer to progressively refine the output token. EE-LLMs introduce mechanisms that enable output tokens to exit at an intermediate layer if a predefined confidence criteria is satisfied. However, we observe that typically, low-confidence tokens which fails to take early-exit, remain unchanged even after additional model traversal. For example, Figure 13 shows the fraction of tokens that are produced by an exit layer which remain unchanged even after traversing the full-model. We observe that in the case of the Llama2-7B (Touvron et al., 2023) model, the eighth layer (one quarter of the model depth) accurately produces upto 90% of the tokens for the the CNN-Dailymail (Nallapati et al., 2016) dataset. Furthermore, this trend remains consistent across models of various sizes from the Llama family (Touvron et al., 2023; Dubey et al., 2024; Roziere et al., 2023), indicating that early-exit layers possess strong predictive capabilities irrespective of model scale. HELIOS leverages this insight to greedily load only the most likely to be used layers, which yields memory savings. These memory savings are then repurposed to support support larger batch sizes.

E Profiling and Storage Overheads

The profiling phase in HELIOS introduces additional computation and storage requirements. However, these overheads are negligible in practice due to the following reasons:

1. **Tokens from the Profiling Step are Retained:** As HELIOS already selects high-quality candidate models, tokens generated during the candidate evaluation or re-assessment phase are not discarded but instead used to fulfill requests.

2. **Latency Overhead:** Loading additional layers or switching to other models happens very infrequently. For example, our studies using a request stream consisting of 1087 requests and two models (OPT-1.3B and OPT-6.7B) show that these transitions occur a mere six times. Moreover, the overhead of these transitions is negligible due to the optimizations described in Section 5.8. The throughput benefits due to maximizing early exits and increasing batch sizes far outweigh these overheads. For example, the total overhead of a profiling iteration in HELIOS when using OPT-1.3B and OPT-6.7B is only 220ms, whereas throughput improves by $1.48\times$ (Section 5) compared to the baseline.

3. **Metadata Footprint:** HELIOS relies on lightweight scalar data structures in its design. For example, the performance history table requires less than 1 KB and the model repository containing offline profile information consumes only a few MBs. These requirements are negligible compared to other memory structures such as model weights and KV caches, which require several hundreds of GBs.

F Evaluated Models

We evaluate HELIOS using both pre-trained and post-training modified EE-LLMs. Pre-trained models are trained to allow intermediate exits directly during the models original training process, while post-training modified models refer to standard off-the-shelf LLMs that are later augmented with early-exits through fine-tuning. The remainder of this section provides specifics for each category:

Pre-trained models: Layerskip (Elhoushi et al., 2024) has publicly released several pre-trained early-exit models on HuggingFace (hug, 2016), spanning the Llama2 (Touvron et al., 2023), Llama3 (Dubey et al., 2024) and CodeLlama (Roziere et al., 2023) families. These models are trained to allow tokens to exit after any arbitrary layer, offering greater flexibility for HELIOS in greedily loading only the most likely to be used model layers. Compared to their standard counterparts, Layerskip variants achieve comparable performance while supporting early-exits.

Post-training modified models: We extend two base models from the OPT family by inserting auxiliary heads at selected intermediate layer depths. These models are then fine-tuned with the backbone parameters kept frozen, while only the parameters of the heads being updated in every iteration. Prior work (Xin et al., 2021; Pan et al., 2024) has observed that such selective fine-tuning preserves the output generation ability of the backbone model while enabling efficient early-exits. The total loss is computed by summing the loss at each exit-layer with a weight of 1.0. For fine-tuning, we utilize the RedPajama (Computer, 2023) and the Pile (Gao et al., 2020) datasets over a total of 50K iterations. Table 9 summarizes the models used in our evaluations:

Table 9. Overview of models used for evaluating HELIOS. Pre-trained models permit tokens to exit at any arbitrary layer, while post-training modified models restrict exits to specific layers.

Family	Parameters	Depth	Early-Exits
`Pre-trained`			
Llama2	7B	32	Anywhere
	13B	40	
	70B	80	
Llama3	8B	32	
CodeLlama	34B	48	
`Post-training modified`			
OPT	1.3B	24	6, 12, 24
	6.7B	32	9, 17, 32

G Additional Results

HELIOS is a unified and flexible framework capable of optimizing for any user-defined SLO. In this section, we show the effectiveness of HELIOS for three SLOs: response-time, accuracy, and energy-efficiency.

G.1 Response Time

When the user's objective is to minimize response time, we evaluate the *Time Taken to First Token or TTFT (lower is better)*, shown in Figure 15. TTFT is limited by the time taken to process all the input tokens in the request. During this time, no output tokens are produced. As expected, larger models, like OPT-6.7B, incur a higher TTFT (69ms) compared to the smaller OPT-1.3B model (43ms). In contrast, HELIOS achieves $1.39\times$ and $2.23\times$ reduction in TTFT compared to OPT-1.3B and OPT-6.7B respectively. This is expected as HELIOS greedily loads only the most likely to be used layers. With HELIOS, each input token in the request traverses fewer layers, significantly reducing the time spent in processing input tokens. Also, HELIOS maximizes the total number of requests served using early exits from both models combined. This is particularly evident in Figure 15 for requests 272 to 542 which corresponds to the CNN-Dailymail dataset comprising long input tokens. HELIOS outperforms the OPT-6.7B model by up to $30\times$ for some of these requests due to reduced layer traversal.

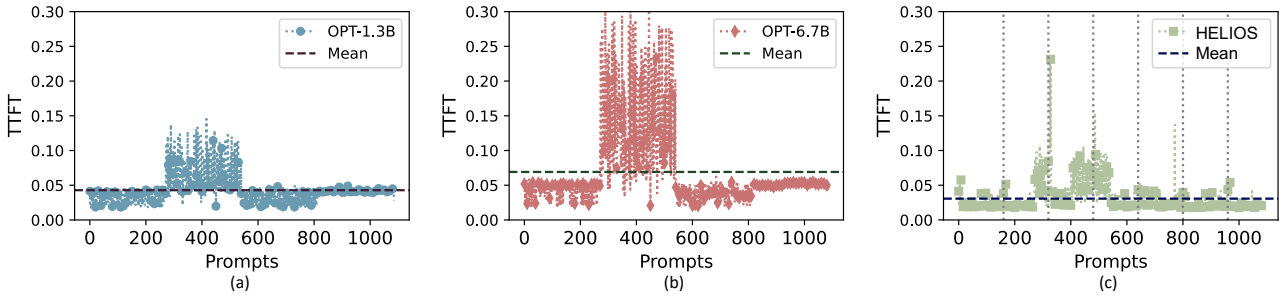

Figure 15. Comparison of *Time To First Token (TTFT)* when using (a) only OPT-1.3B, (b) only OPT-6.7B, and (c) HELIOS (*lower is better*). In (c), the vertical lines denote timestamps when a candidate re-assessment is initiated.

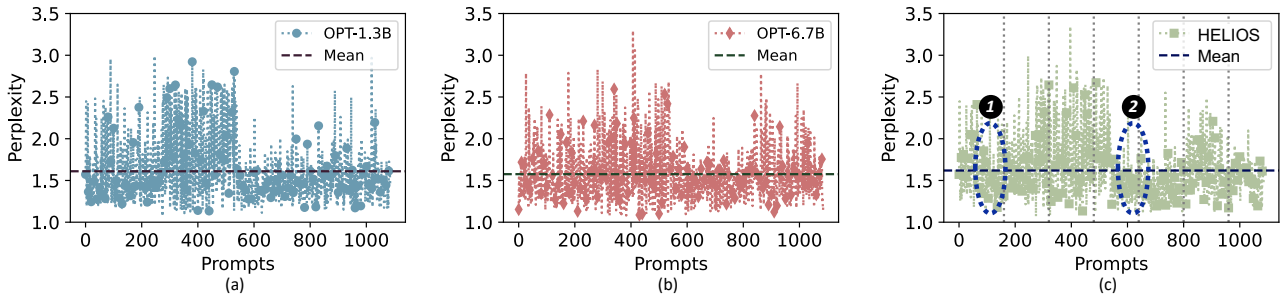

Figure 16. Comparison of *Perplexity (Accuracy)* when using (a) only OPT-1.3B, (b) only OPT-6.7B, and (c) HELIOS (*lower is better*). Vertical lines represent timesteps when a candidate re-assessment is initiated.

G.2 SLO: Accuracy Optimization

Next, we study the scenario when the user's goal is to maximize the accuracy. Figure 16 shows the *perplexity (lower is better)* for three model selection cases. The perplexity of the larger model, or the OPT-6.7B model, is lower ($0.97\times$) than the smaller OPT-1.3B model. This is expected because larger models typically have many parameters enabling them to better capture the subtle relationships between tokens and produce more nuanced outputs. Specifically, for long context benchmarks, where a significantly large number of output tokens are generated, larger and complex models are known (Achiam et al., 2023; Kaplan et al., 2020) to outperform smaller models. Hence, for the set of prompts corresponding to CNN Daily Mail (Nallapati et al., 2016), which comprises relatively long context inputs, HELIOS switches to using OPT-6.7B in real-time, as illustrated by ❶ in Figure 16(c), to meet the target SLO of the user. This highlights the ability of HELIOS to adapt to various SLO requirements. On the other hand, HELIOS reverts back to OPT-1.3B later, as shown by ❷ in Figure 16(c), because HELIOS evaluates that it offers accuracy comparable to OPT-6.7B in a more energy-efficient manner (with fewer layers and reduced computational footprint).

G.3 SLO: Energy-efficiency Optimization

We briefly discuss the scenario when a user wants to maximize the energy-efficiency or *minimize the energy per prompt*. Using only OPT-6.7B consumes 1.01 Wh of energy per prompt which is expected given it is a larger model compared to OPT-1.3B that consumes 0.50 Wh per prompt. In contrast, HELIOS consumes 0.45 Wh of energy per prompt, which translates to 10% energy savings, for comparable perplexity. In HELIOS, 58.3% of the prompts are serviced using partially loaded models , which yields the observed energy savings. Note that savings scale with the total number of prompts processed. In practice, production servers in datacenters process tens of millions of prompts daily (Wang et al., 2023), emphasizing the impact of HELIOS. We also observe that the energy overheads associated with switching is minimal, comprising only $0.05\times$ of the overall energy savings (10%) achieved.