HELIOS: Adaptive Model And Early-Exit Selection for Efficient LLM Inference Serving

Avinash Kumar
avinkumar@utexas.edu
The University of Texas at Austin
Austin, TX, USA

Shashank Nag
shashanknag@utexas.edu
The University of Texas at Austin
Austin, TX, USA

Jason Clemons
jclemons@nvidia.com
NVIDIA
Austin, TX, USA

Lizy John
ljohn@ece.utexas.edu
The University of Texas at Austin
Austin, TX, USA

Poulami Das
poulami.das@utexas.edu
The University of Texas at Austin
Austin, TX, USA

Abstract

Deploying large language models (LLMs) presents critical challenges due to the inherent trade-offs associated with key performance metrics, such as latency, accuracy, and throughput. Typically, gains in one metric is accompanied with degradation in others. *Early-Exit LLMs (EE-LLMs)* efficiently navigate this trade-off space by skipping some of the later model layers when it confidently finds an output token early, thus reducing latency without impacting accuracy. However, as the early exits taken depend on the task and are unknown apriori to request processing, EE-LLMs conservatively load the entire model, limiting resource savings and throughput. Also, current frameworks statically select a model for a user task, limiting our ability to adapt to changing nature of the input queries.

We propose *HELIOS* to address these challenges. *First*, HELIOS shortlists a set of candidate LLMs, evaluates them using a subset of prompts, gathering telemetry data in real-time. *Second*, HELIOS uses the early exit data from these evaluations to greedily load the selected model only up to a limited number of layers. This approach yields memory savings which enables us to process more requests at the same time, thereby improving throughput. *Third*, HELIOS monitors and periodically reassesses the performance of the candidate LLMs and if needed, switches to another model that can service incoming queries more efficiently (such as using fewer layers without lowering accuracy). Our evaluations show that HELIOS achieves 1.48× throughput, 1.10× energy-efficiency, 1.39× lower response time, and 3.7× improvements in inference batch sizes compared to the baseline, when optimizing for the respective service level objectives.

1 Introduction

Large Language Models (LLMs) are becoming increasingly prevalent due to their ability to perform a variety of tasks, such as content creation, summarization, etc. [1–3]. However, deploying LLMs present critical challenges due to the inherent trade-offs that exist across key performance metrics, like *accuracy*, *throughput*, and *latency*. For example, large and complex LLMs offer high accuracy at the expense of increased latencies and reduced throughput. Consequently, various approaches, such as advanced memory management [4–9], enhanced request handling [10, 11], compute reduction [12–16], are being developed to efficiently navigate these trade-offs. This paper focuses on *Early-Exit LLMs (EE-LLMs)*. EE-LLMs adapt the computational cost of producing outputs based on the complexity of the prompt, with simpler prompts using fewer computations than the complex ones.

Why Early-Exit Models (EE-LLMs)? EE-LLMs exploit the insight that not all prompts are equally complex and the first few model layers are often sufficient for most trivial prompts. EE-LLMs introduce exit paths in the model, and if a token is identified with a probability greater than a *confidence threshold*, the remaining layers are skipped. The computational savings, latency reduction, and throughput improvements scale proportionately with the number of layers skipped. This makes EE-LLMs attractive because they improve latency and throughput without degrading accuracy. Consequently, EE-LLMs have been widely adopted in various deep learning architectures in both industry and academia [17–25].

Limitations of Current EE-LLM Serving: Contemporary frameworks [26–35] *statically select a model* to meet user-specified service-level objectives (SLOs) based on generic telemetry data (such as accuracy, latency, etc.), which often fails to capture the specific nuances of the input task, leading to sub-optimal model selection. Our studies show that even naively choosing the largest model is not always optimal. Moreover, two models with the same number of parameters offer different accuracy versus latency trade-offs. For example, Llama-7B [36] and Mistral-7B [37] have comparable accuracy for the TruthfulQA dataset [38], but Mistral-7B requires 0.90× latency and consumes 0.85× energy compared to Llama-7B. Static model selection also limits us from adapting to the changing characteristics of the incoming queries.

Moreover, EE-LLMs also offer only *limited resource savings and throughput improvements*, despite significantly reducing computational costs. This stems from the non-deterministic nature of early exits taken, which depends on the input task

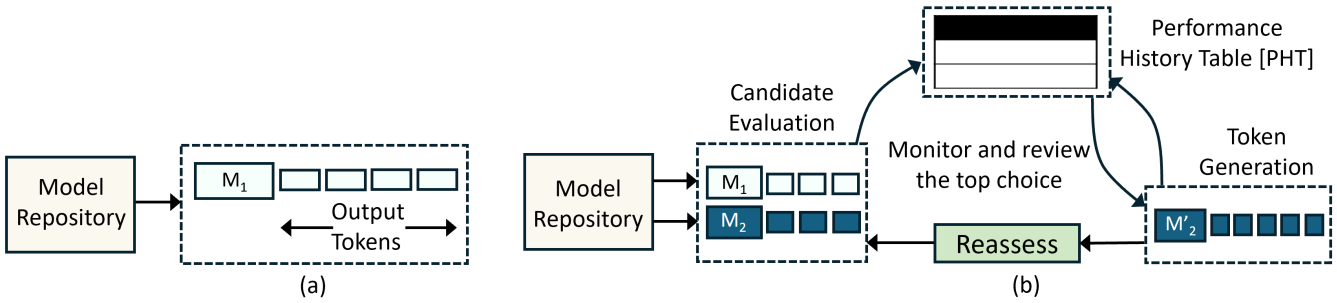


Figure 1. (a) Today, we statically select an LLM from a repository (M_1 in this illustration) and generate all output tokens using it. (b) HELIOS selects multiple LLMs (M_1 and M_2 here), evaluates them by generating a few prompts from the input task, picks the best one, and loads the layers most likely to be used based on early exits taken during evaluation (M_2'). HELIOS also monitors the performance in real-time and re-evaluates the models and switches to another model if needed.

and are only known at runtime. Also, not all output tokens take early exits. Our studies using a prompt mix and the OPT-1.3B model with 24 layers show that although 74% of the tokens exit after layer 6, 5% of the tokens exit only after layer 12, and 21% of the tokens require all 24 layers. To cater to all requests, current solutions conservatively load the entire model (weights for all layers), yielding practically no memory savings. EE-LLMs also offer limited throughput gains due to poor support for *batched inference*. Typically, multiple requests are "batched" and processed simultaneously to improve throughput. In such scenarios, a token is generated for every request, before proceeding to produce the next token for all requests. To process a batch of requests, EE-LLMs must therefore, either wait for the token that takes the maximum number of layers to complete or naively adopt a batch size of 1. Given the synchronization overheads involved in the former, existing EE-LLMs use the latter by default [39, 40].

Our Proposal: To efficiently serve EE-LLMs, we propose *HELIOS* that leverages two key insights. *First*, HELIOS selects the optimal model by dynamically collecting telemetry data for a given task instead of relying on similar data from generic benchmarks. However, this is non-trivial due to the availability of a large number of LLMs. To address this, HELIOS uses the benchmark-based telemetry data to first narrow down the search space to a set of candidate models that meet the target SLOs. Next, HELIOS *evaluates* each candidate model by using them to generate output tokens for a limited number of input prompts. This evaluation is bound to give more accurate telemetry data because it (1) caters to the specific task and (2) accounts for early exits taken while processing requests. HELIOS saves this information in a *Performance History Table (PHT)* and selects the best-performing model for further token generation.

Second, our experiments show that even if the confidence threshold is not met at an early-exit, the predicted token often remains *unchanged* even after traversing through additional layers. Our studies using the cnn-dailymail dataset [41] and Facebook OPT-6.7B model show that there is a 92.1% chance that tokens restricted from taking the first exit (Layer-9) due to a low confidence score of 0.2, ultimately become the final output after traversing through all layers. Building on this insight, HELIOS uses early exit data from the PHT to greedily load only a limited number of layers for the chosen model. This approach in HELIOS yields memory savings which can be eventually repurposed to support *large batch sizes* and serve multiple requests concurrently. Moreover, it ensures that each token generated only traverses a fixed number of model layers, eliminating synchronization overheads between input prompts within a batch.

Although this approach suffices for most requests, HELIOS seldom encounters tokens that do not meet the confidence threshold. HELIOS has two options to handle such cases-either (1) load all layers of the same model currently in use or (2) switch to an alternate model that can potentially complete the task in a more efficient manner by using early exits. In either case, model weights must be loaded onto GPU memory (corresponding to the extra layers of the current model or fewer layers of an alternate model). To reduce these overheads, HELIOS exploits two insights. *First*, it evaluates the overheads of each option by using the PHT and selects the one meeting target SLOs with minimal overheads. *Second*, HELIOS makes this decision only if a certain number of tokens within a window of consecutive tokens do not meet the confidence threshold as processing additional layers often only improves the confidence, while the predicted token itself remains unchanged.

Figure 1 illustrates prompt processing in existing frameworks versus HELIOS. The evaluation phase uses models M_1 and M_2 to service the first few input prompts before HELIOS decides to select M_2 for further token generation. M_2' denotes model M_2 loaded up to a subset of layers. In the event M_2' does not meet the confidence thresholds and it is identified that M_1' (model M_1 loaded up to select exit layers) is a better candidate, further tokens are generated using M_1'.

Our evaluations show that HELIOS achieves 1.48× throughput, 1.10× energy efficiency, and 1.39× lower response time while maintaining comparable accuracy to the baseline static model selection policy, along with an average 3.7× improvement in batch size, when optimizing for the respective SLOs.

Overall, this paper makes the following contributions.

1. We show that the performance of EE-LLMs depends on the nature of input queries and propose *HELIOS* that enables dynamic model selection tailored to each incoming request.
2. HELIOS observes performance and greedily loads only a subset of the selected model, yielding memory savings which is used to enable batched-inference and improve throughput.
3. To maintain accuracy while operating with fewer layers and adapt to the changing nature of input queries, HELIOS monitors the performance in real-time and switches to another model or loads the remaining layers of the current model based on the performance trade-offs, while exploiting prompt locality, and optimizing for user-specified SLOs.

2 Background and Motivation

2.1 Large Language Models

Large Language Models (LLMs) are very attractive machine learning algorithms due to their ability to make meaningful predictions using a relatively small number of prompts or inputs and flexibility to perform a variety of tasks such as text generation, code creation, recommendations, and classification. [1–3]. The efficacy of an LLM is measured by its inference *accuracy*, *throughput*, and *latency*. However, there exists inherent trade-offs in these metrics, where improving one often degrades another. For instance, larger and complex models capture more subtle patterns in the input prompt, improving accuracies. But this also comes at the cost of increased latencies and reduced throughput due to higher computational costs. For example, as shown in Figure 2, the accuracy of `openbookqa` [42] increases from 26.4% to 35.2% as the number of parameters and computations increase by 13× and 8× respectively for Llama models.

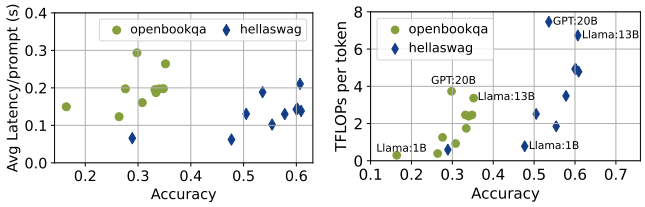


Figure 2. (a) Average latency (in seconds) and (b) computational complexity (in TeraFLOPs) per token versus accuracy for openbookqa and hellaswag for different models.

2.2 Servicing an Input Request Using an LLM

Figure 3 shows the steps involved in processing an input request with an LLM. It starts with *model selection* for the task, after which its pre-trained weights are loaded onto GPU memory. The LLM then parses the contents of the input prompt. This is called the *Prefill* phase. It is followed by the *Token Generation* phase, in which multiple output tokens are produced one by one in an auto-regressive manner, where each token generated depends on all previously generated tokens for contextual information. The end of sequence (EOS) token concludes the decoding phase and marks the completion of the service request.

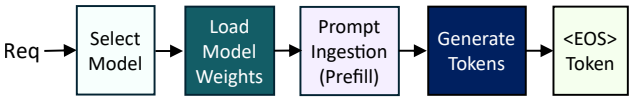


Figure 3. Steps in processing an input request using an LLM.

2.3 Early Exit Large Language Models

Early-Exit LLMs or *EE-LLMs* are a variant of LLMs that improve throughput and latency without sacrificing accuracy [12, 43–45]. While traditional LLMs traverse all decoder layers to generate tokens, EE-LLMs exploit the insight that not all prompts are complex and going through additional decoder layers for trivial prompts have diminishing returns. For example, in Figure 2, the accuracy of `hellaswag` [46] increases from 60% with Llama-3-8B (32 layers and 8B parameters) to only 61% with Llama-2-13B (40 layers and 13B parameters), while increasing latency by 1.47×. For simple queries, whenever EE-LLMs reach a layer with both exit and forward paths, they exit if a token is identified with a probability (P) above a threshold (TH), as shown in Figure 4.

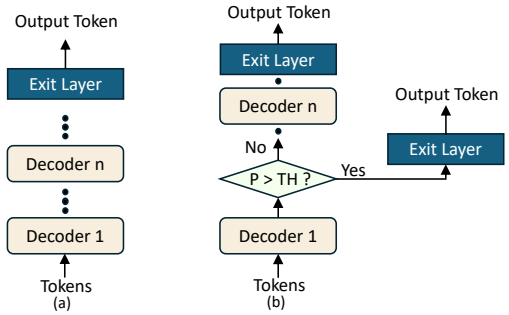


Figure 4. (a) Transformer architecture in LLMs. (b) In Early-Exit LLMs, each token generation takes one of the early exits, or the final one, depending on a confidence threshold (TH).

2.4 Drawbacks of Existing EE-LLM Serving

Existing EE-LLM serving techniques suffer from three key drawbacks that limit their efficiency and performance. First,

they statically select a model and generate all output tokens using this model. Second, early exits taken are task-dependent and unknown prior to servicing requests. More importantly, not all tokens take early exits. Consequently, existing methods are forced to load the entire model. Third, they offer limited compatibility with other widely prevalent throughput enhancement techniques such as *batching*. Batched inference generates output tokens in lockstep, with one token being generated per request in the batch at each generation step before proceeding to the next step. This introduces synchronization overheads and the time taken to generate a token is limited by the token that takes the longest to exit. These limitations severely hinder us from efficiently navigating the trade-offs across the key performance metrics.

2.4.1 Static Model Selection Is Sub-Optimal.

Identifying the best model for a task is non-trivial because it depends on the nature of the task, which is unknown. Figure 5 shows the accuracy of two datasets on ten models. We observe that naively selecting the largest model, Llama2-13B, for `gsm8k_cot` [47] is sub-optimal. It has 37% lower accuracy than the best model, Llama-3.1-8B, while incurring 1.9× latency and 2× energy consumption. In contrast, Llama2-13B offers 20% higher accuracy, albeit with 1.4× latency and 1.6× energy for `ethics_justice` [48]. Our studies show that even the same model, fine-tuned for different tasks, offers different accuracy versus latency trade-offs.

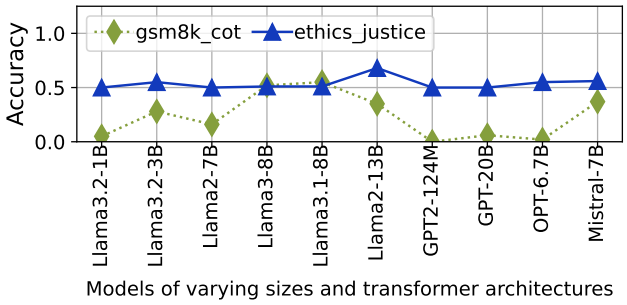
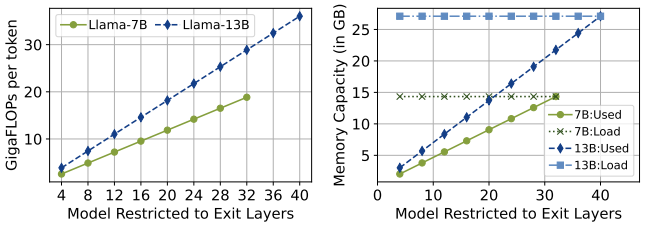


Figure 5. Accuracy for two datasets on ten models. It depends on the model parameters, architecture, and input task, making it harder to select the best model for unknown tasks.

Model selection depends on service level objectives (SLOs) of users. Cost-focused SLOs reduce resource usage (such as power and memory) while maximizing throughput, whereas performance-oriented SLOs minimize latency within a power budget. Prior work InFAAS [26] uses knowledge about the task (image classification, code generation, etc.) to choose a model variant and hardware deployment strategy to meet the SLOs. But, once selected, all requests are served with the same model configuration. Thus, InFAAS cannot account for variations in input queries. Clipper [27], uses multiple models and aggregates their predictions for a given prompt. Although it captures the performance trade-offs across prompts, Clipper is extremely resource-intensive.

2.4.2 Exits Unaware Loading limits Memory-Savings.

Although EE-LLMs reduce computational requirements, their memory footprint is identical to the entire model. Figure 6(a) shows the computational savings for Llama-7B (32 layers) and Llama-13B (40 layers) depending on the exit layer, exiting after 16 layers yields 50% and 60% computational savings respectively. Figure 6(b) compares the memory capacity accessed for these models. Notably, if the model is always known to exit within the first k layers, selectively loading only those layers would reduce the overall memory footprint.



(a) (b)

Figure 6. Resource estimates of two Llama models (7B and 13B). (a) GigaFLOPs per token versus exit layer- earlier exits yield computational savings. (b) Memory capacity versus exit layer- earlier exits do not access the entire memory but the total footprint is unchanged as the entire model is loaded.

2.4.3 Exits Taken in EE-LLMs Are Unknown.

Loading the optimal model, wherein we only load the weights corresponding to the layers that will be used is challenging because the early exit taken depends on the task and is known only at runtime. Figure 7 shows that over 70% of tokens generated use only up to 25% and 28% of the total layers of OPT-1.3B and OPT-6.7B models respectively for a mixture of prompts. Although ideally, we could only load up to layers 6 and 9 respectively for most output tokens, in practice this information is unknown and existing EE-LLMs conservatively load the entire model. More importantly, not all tokens exit early and additional model layers are still required to service about 20% of the tokens.

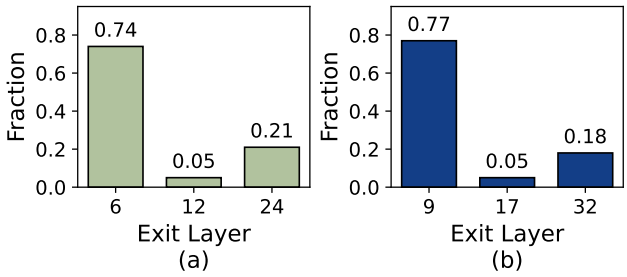


Figure 7. Distribution of exit layers taken by tokens for the same prompt mix with OPT- (a) 1.3B and (b) 6.7B models.

3 Our Proposal: HELIOS

In this paper, we propose *HELIOS*, a framework that efficiently serves EE-LLMs by dynamically selecting the optimal model based on input-specific telemetry data. By greedily loading only the most likely to be used layers, HELIOS reduces memory usage and computational overheads. This approach enables HELIOS to adapt to the changes in input queries, while also improving throughput.

3.1 Key Insights

HELIOS leverages the following key insights.

3.1.1 Is Meeting Confidence Always Necessary?

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 decoder layers. For example, Figure 8 shows the fraction of tokens which remain the same even after traversing additional decoder layers depending on the confidence threshold required to take an early exit at Layer-9. If we set a confidence threshold of 1.0, none of the tokens would take the first exit. However, we observe that in 85% of the cases, the token predicted at the first exit and the final exit is the same – with subsequent layers only improving the confidence. A prior work [49] also makes a similar observation.

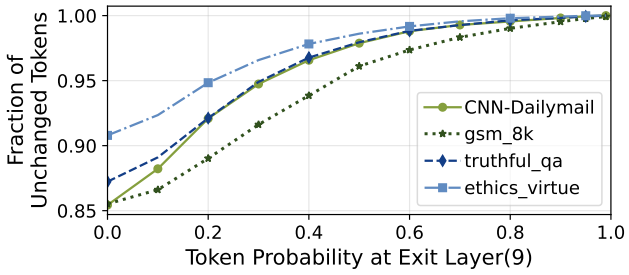


Figure 8. Fraction of unchanged tokens for four datasets on OPT-6.7B model from the 1st exit layer (9) to the final layer (32). We observe that probability of the predicted token staying unchanged is always greater than 85%

Insight-1: Not meeting confidence is okay at times

As most low confidence tokens remain unchanged after processing additional layers, greedily exiting early does not significantly impact accuracy.

3.1.2 How are Early Exits Distributed Across LLMs?

Our studies show that often, tokens requiring all layers of one model (i.e. no early exits taken) can be predicted accurately with another model using an earlier exit. Figure 9 shows that early exits depend on the EE-LLM. Prompts needing more than layers on OPT-1.3B require fewer layers on OPT-6.7B for similar accuracy. We make similar observations for prompts needing more than layers on OPT-6.7B.

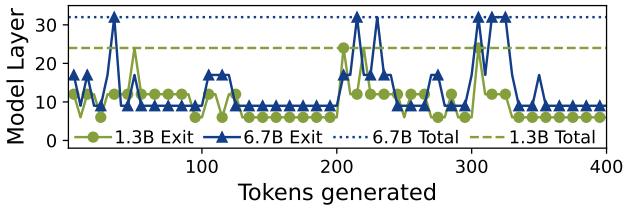


Figure 9. Exit layers for serving a typical workload with a mixture of prompts using OPT 1.3B and 6.7B models.

Insight-2: Early exits are often complimentary

Tokens requiring more layers in one model can be predicted accurately using fewer layers in another model.

3.1.3 Prompts Exhibit Locality.

Even the most advanced LLM rarely gives the intended response on the first try [50–52]. Consequently, users typically submit multiple queries with incremental adjustments to steer the LLM towards providing more nuanced responses. Thus, these subsequent prompts often have overlapping contexts, exhibiting locality. In fact, this attribute is often used to improve caching strategies in LLMs [53] and several inference engines, including TensorRT-LLM [54], AWS [55], chatGPT [56], integrate this optimization. Additionally, there exists dedicated efforts to engineer prompts to exhibit greater degrees of locality [57–61]. Crafting an effective prompt involves carefully formulating queries to guide the LLM in producing specific, accurate, and relevant outputs based on the LLMs initial response. For example, instead of broad queries such as "Summarize the attached document?", a more precise prompt would be "Highlight the key insights in the attached document?". This process of iteratively refining prompts leads to repeated patterns in the input queries processed by the LLM, thereby exhibiting locality.

Insight-3: Recognizing the optimal model and early exits for some tokens is enough

As prompts exhibit locality, the selected model and early-exits for a subset of prompts are likely to maintain high performance even for subsequent input requests.

A key limitation of existing EE-LLM serving frameworks is the reliance on a statically selected model that also lack mechanisms to evaluate the trade-offs associated with early exits. This limits us from exploiting the insights discussed above in real-time on a per task basis. Our proposed solution, HELIOS is a software framework that addresses these limitations and enables dynamic model selection and efficient usage of EE-LLMs tailored to each task.

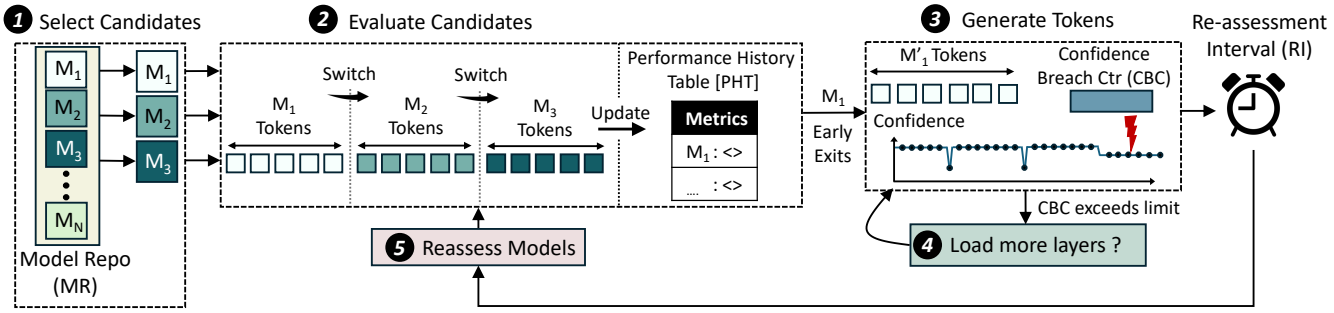


Figure 10. Design of HELIOS

3.2 Design Overview

Figure 10 shows an overview of HELIOS. ❶ HELIOS selects a set of potential candidate models and ❷ evaluates their performance in real-time before converging on using one of them. 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 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 input tasks.

3.3 Design Implementation

Next, we discuss the implementation of HELIOS.

3.3.1 Step-1: Selection of Candidate Models.

Recent advancements have led to the availability of a large number of LLMs that continue to grow at an unprecedented rate. For example, Hugging Face [62] hosts more than 150,000 LLMs today. This large search space imposes a significant challenge in identifying the most appropriate model for an incoming task. To reduce the search space, HELIOS selects *TopK* candidate models based on telemetry data collected from standard benchmarks [38, 42, 46, 63–70] (similar to static model selection). HELIOS maintains a *Model Repository (MR)* that holds relevant telemetry data such as throughput, accuracy, latency and memory footprint for a comprehensive set of benchmarks and a large set of models. The MR uses benchmarks offering a good coverage of tasks handled by LLMs and ranges from text summarization [71, 72] to code generation [73, 74], grade-school math [75, 76], language translation [77, 78], ethics [79, 80], etc. For example, in Figure 10, models M_1, M_2, and M_3 are selected as candidate models. The default implementation of HELIOS selects two models in this step to restrict the complexity of our evaluations on limited hardware resources (GPUs).

3.4 Step-2: Evaluation of Candidate Models

Next, HELIOS evaluates the performance of the selected candidate models in real-time before using one of them. This approach is effective because successive queries often share *overlapping contexts* and exhibit locality (***Insight-#3***). For the evaluations, HELIOS sequentially generates output tokens using candidate model for a subset of input prompts. By default, HELIOS evaluates each candidate for five prompts. Figure 10 illustrates this, where models M_1, M_2, and M_3 generates output tokens for five prompts. Note that although we could evaluate all candidate models in parallel, it is too resource-intensive. We avoid this mainly due to the limitations of our setup (not enough GPUs) and reduce wasted energy consumption. Moreover, this does not impact performance, 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. Consequently, the time taken to generate the first token (*TTFT*), which measures response time and is a key performance metric for LLMs, is not impacted.

Model Evaluation Methodology: The throughput, latency, early exits, and energy are measured using profiling tools. Assessing accuracy is non-trivial because we lack the ground truth for comparison. To address this challenge, we use reference free metrics such as perplexity [81–83], Supert [84], Rouge-C [85], and BLANC [86]. Once candidate evaluation completes, a model is chosen to meet user-specified SLOs. For example- if the SLO is to maximize accuracy, HELIOS selects the model the lowest perplexity from the candidate pool. Next, the chosen model and its early exit distribution are sent to the next stage. Table 1 shows how HELIOS can enhance model selection via the candidate evaluation step. It shows the perplexity of two different models. If we were to statically select a model, based on the benchmark-based telemetry data, we would have selected the OPT-6.7B model. However, the post-evaluation data shows the OPT-1.3B model to be more effective for the current set of prompts. HELIOS uses it to process the incoming requests further.

Performance History Table: The history of the key performance metrics (throughput, latency, accuracy, and energy)

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

as well as the early exit distribution of each model is saved in a table, called the *Performance History Table (PHT)*. The PHT is used in the next stages of HELIOS, as discussed next.

3.4.1 Step-3: Token Generation Using Best Model.

The model identified as the best candidate or "optimal" in the evaluation stage is now employed to generate the remaining tokens for the pending requests.

Greedy Loading Up to Selected Exit Layers: Loading only up to the exit layers where most tokens are likely to exit, yields memory savings which can now be used to process additional requests (increased batch sizes). Unlike current EE-LLM serving frameworks, HELIOS has access to the early exit history from the evaluation phase saved in the PHT. For example, Figure 11(a) shows the exits taken for a prompt mix using the OPT-1.3B model. We observe that 74% of the requests only require 6 layers of the model. We refer to these as *Low-Exit Tokens (LTs)*. HELIOS greedily loads only up to 6 layers in this scenario (denoted by M_1' in Figure 10). If most pending requests are LTs that do not require additional layers, this greedy approach yields significant memory and energy savings, without compromising accuracy.

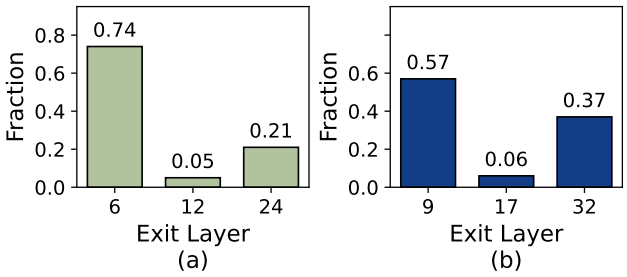


Figure 11. Distribution of exit layers for tokens of a prompt mix using the OPT-1.3B Model. 74% of the requests only require up to 6 layers. (b) Distribution of exit layers of the remaining 26% when they are serviced by OPT-6.7B model.

Load More Layers Or Another Candidate Model? Although partially loaded models offer significant resource savings and have the potential to improve larger batch sizes, we encounter tokens where the confidence threshold is not met. For example, in Figure 11(a), 26% of the requests use more than the 6 layers that are loaded. 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 where the same request can be serviced 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 (based on ***Insight-#2***). Figure 11(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 9 layers of the OPT-6.7B model. Note that the exit history information is available in the PHT from the candidate evaluation phase. Loading more layers of the current model is beneficial 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 9 layers in the example). Once the overheads of both options are evaluated, the option with minimal overheads is selected and appropriate action is taken.

Amortizing Loading Overheads with CBC: Irrespective of the option selected, both approaches- loading additional layers and model switching, incur huge overheads. To minimize switching overheads, HELIOS considers both options only after a certain number of tokens within a time window fail to meet the confidence threshold. This leverages our observation that not every token that does not meet the confidence threshold at an early exit layer actually changes in future layers (***Insight-#1***). In other words, even though we do not meet the confidence threshold at an early layer, the output token itself does not change as additional layers are processed, but only its probabilities change and the confidence improves. HELIOS uses a *Confidence Breach Counter (CBC)* which is initialized to 0. The CBC increments each time the confidence threshold is not met for an output token. If the value of the CBC exceeds a pre-determined maximum allowable limit (CBC_{max}) for a window of consecutive tokens, HELIOS decides whether it should load more layers or switch to another model. The CBC is reset after a decision is made. The default implementation of HELIOS only tolerates up to 50 confidence threshold breaches ($CBC_{max} = 50$) in a window of 100 consecutive token.

3.4.2 Step-4: Periodic Re-assessment of Models.

Even though input request exhibit temporal locality, the nature of input tasks often vary over prolonged time periods. For example, the input requests may switch from question-answering to text summarization. It is important to quickly identify such transitions and adapt the model to suit the newer tasks. In the example above, once the task changes to text summarization, the model selected earlier for question-answering could be potentially sub-optimal and a model appropriate for summarization should be chosen. Unfortunately, these transitions in input tasks are unknown and if we wait for too long before we switch to another model, the performance could be impacted. This is because we continued to service requests using the current (potentially sub-optimal)

model. Ideally, we should evaluate on a per-prompt basis if the current model is the most suitable one or a different model is required. However, doing this for every prompt incurs overheads in loading the model weights of other potential candidate models onto the GPU memory and evaluating their performance. To achieve a sweet-spot between switching too frequently and waiting too long, HELIOS reassesses models every 150 prompts by default. We refer to this as the *Re-assessment Interval (RI)*.

Selecting a Re-assessment Interval: As server environments have ample resources, it is possible to amortize the cost of loading alternate models by overlapping the token generation phase of the current model on one server with model loading concurrently on another server. Nonetheless, we still avoid too frequent evaluations regarding model switching (such as evaluating for every prompt) to keep energy consumption within limits. By default, HELIOS assumes a resource constrained environment and uses a simple algorithm that considers the token generation latency of the candidate models and model loading costs to determine the frequency of candidate model re-assessment. We recommend treating the re-assessment period as a hyper-parameter that can be adjusted based on resource availability, and ideally fine-tuned by the inference server provider to suit the specific nature of the tasks being served.

Re-assessment Phase. The model re-assessment step evaluates the candidate models from Step-1 again and updates the PHT. If another model offers better performance, it is loaded for further processing (moving to Step-3). Our default implementation of HELIOS only re-assesses the performance of the previously shortlisted model candidates because our evaluation shows that typically, some models generally outperform for a wide variety of tasks. For example, Figure 12 shows the accuracy of various datasets relative to the best performing model for multiple models. We observe that Llama-3-8B [87, 88], Llama-2-13B [36], and Mistral-7B [37] consistently perform well, whereas GPT2-124M [89] is consistently poor. Nonetheless, HELIOS can also look up the MR to select other candidate models if needed. The steps for HELIOS are summarized in Algorithm 1 in Appendix.

4 Evaluation Methodology

We discuss the methodology used to evaluate HELIOS.

4.1 Models

We consider two models of varying sizes from Facebook's OPT family [90] with 1.3 and 6.7 billion parameters. We add early exits to these models at 1/4th the depth [39, 40]. Prior work EE-Tuning [40] notes that exits at shallower depths, such as 1/4th the model depth, enable faster inference while maintaining performance. Table 2 summarizes the models. The open-source implementations of the models in Table 2 does not include early exits. So, we fine-tune them to calibrate the early exits. To ensure generalization across tasks, we use the red-pajama [91] and pile [92] datasets. For fine-tuning, we use a weight of 1.0 to each early-exit loss in the total loss calculation and use 50,000 iterations.

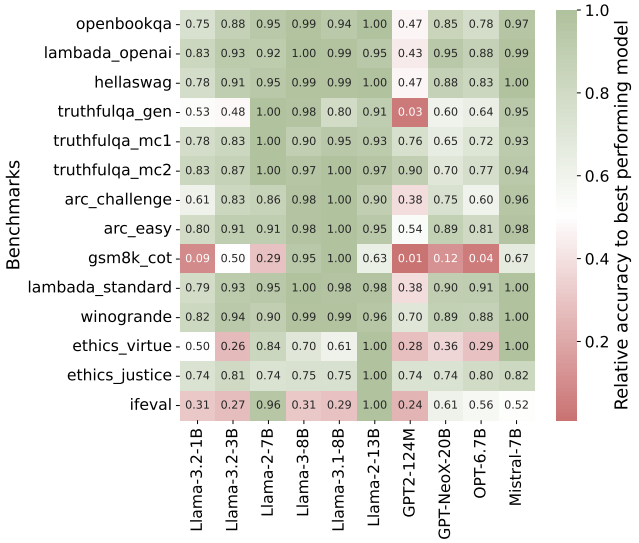


Figure 12. Accuracy of datasets across multiple models relative to the best performing model. Some models consistently perform well while others are consistently poor.

Table 2. Summary of Models Used

Model	Parameters	Layers	Early Exits
OPT-1.3B	1.3 billion	24	6, 12, 24
OPT-6.7B	6.7 billion	32	9, 17, 32

4.2 Setup

We use the EE-LLM framework [39] for both fine-tuning and inference serving, as in prior works [39, 40]. We limit our evaluations to two models due to limited access to GPUs. We use an NVIDIA cluster with 4xA100 (40GB) GPUs and an AMD EPYC 7742 CPU with 64 cores. An NVSwitch connects the GPUs at a 600 GB/s inter-GPU bandwidth. We use tensor parallelism [93] of 1 and pipeline parallelism [94] of 4.

4.3 Datasets

To mimic the dynamic trade-offs across model variants with varying nature of prompts, we consider a mix of requests presented to the inference server for evaluation. This is composed of prompts from standard benchmarks [63, 95–100] such as CNN Daily Mail [41], TruthfulQA [38], OpenBookQA [42], gsm8k[101], code generation [102] and sentence completion [46]. For each prompt, we restrict the generated output to 100 tokens to avoid running out of memory on GPUs, before serving the next prompt.

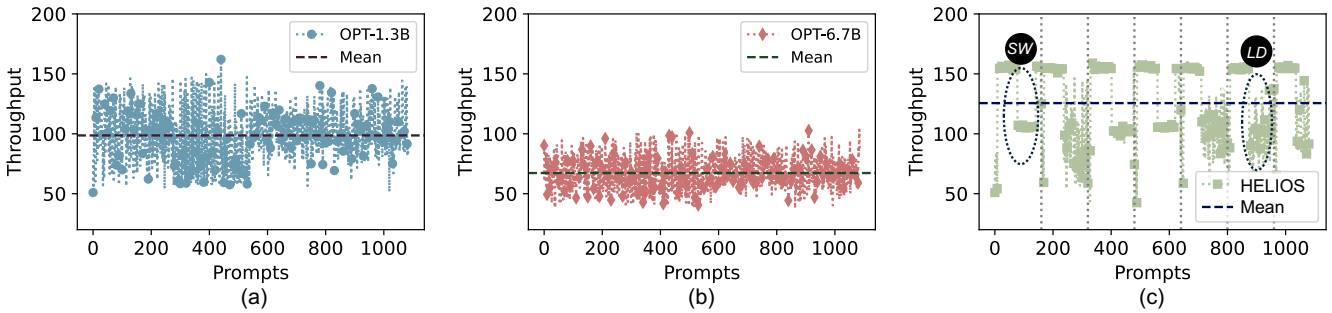


Figure 13. Comparison of *Throughput* when using (a) only OPT-1.3B, (b) only OPT-6.7B, and (c) HELIOS (*higher is better*). Candidate re-assessment steps are denoted using vertical dotted lines in (c).

4.4 Figure-of-Merit

We use perplexity to evaluate the accuracy of the generated tokens to be consistent with prior works [81, 103–105]. Perplexity is a reference-free accuracy metric used for LLM evaluation. Additionally, we use NVIDIA-SMI to measure power and memory footprint. Our studies show NVIDIA-SMI samples reliably at the granularity of 1 second. We post-process this data to measure performance (such as throughput, etc). Table 3 summarizes the metrics used.

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) × Number of Tokens
Throughput	Tokens generated per second ($\frac{1}{TPOT}$)

We evaluate HELIOS assuming four user-specified SLOs-throughput, response time, accuracy, and energy-efficiency.

5 Results

5.1 SLO: Throughput Optimization

We consider the user's SLO is to maximize throughput. We use a batch size of 1 as used in existing EE-LLM serving [39, 40]. 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) use smaller models for early exits because traversing a layer takes longer on larger models. Figure 13 shows the *throughput (higher is better)*. About 91% of the tokens are processed using the earliest exits of both models combined (layer 6 of OPT-1.3B and layer 9 of OPT-6.7B). Also, a significant portion (77%) of these tokens are processed using 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 model standalone. Consequently, HELIOS improves the throughput by 1.48× and 2.13× on average compared to using OPT-1.3B and OPT-6.7B standalone respectively. In Figure 13(c), *LD* shows scenarios where more layers of the current model are loaded, whereas *SW* denotes cases where HELIOS switches to another model, highlighting HELIOS's adaptive nature that balance trade-offs between key performance metrics. We also note that by optimizing for throughput, HELIOS does not significantly impact the other performance metrics – infact, energy per inference improves by 1.06× over the OPT-1.3B model and 2.17× over the OPT-6.7B model, and the model perplexity is only 0.01 less than the OPT-1.3B model.

5.2 Enabling Larger Batch Sizes in EE-LLMs

HELIOS supports larger batch sizes in EE-LLMs by (1) eliminating synchronization overheads and (2) re-purposing the GPU memory saved by not loading all model layers to service additional requests. In HELIOS, all tokens at any given token generation timestep must exit the same early exit layer (up to which the 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 process tokens. While model weights can be shared, each request must maintain its own KV-caches. Loading only a subset of layers yields considerable memory savings in HELIOS, as shown in Table 4. Moreover, as HELIOS uses fewer layers, the KV cache sizes required also reduces proportionately. There also exists methods that optimize KV caches independently [106–109]. HELIOS is orthogonal to these works and both can be combined together for even greater benefits.

Table 4. Nvidia A100 memory savings by loading up to the first exit layer across different models.

Model (Layers)	Exit Layer	Memory Savings (%)
OPT-1.3B (24)	6	4.6
Llama2-7B (32)	9	22.5
Llama2-13B (32)	15	37.8

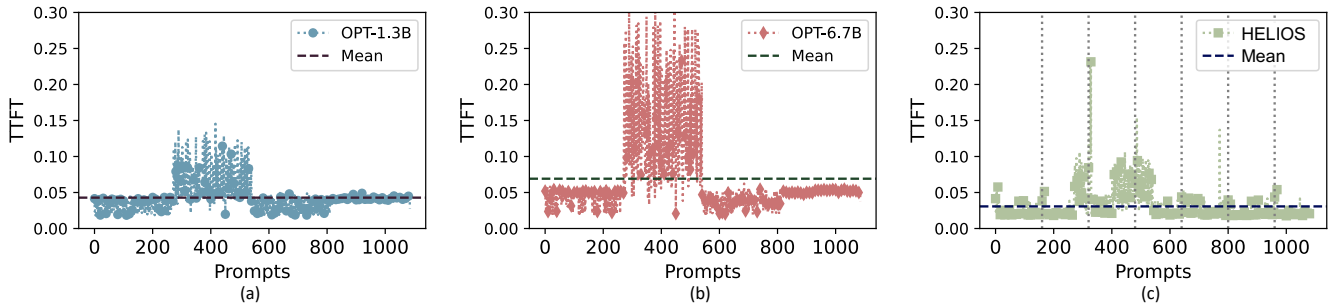


Figure 14. Comparison of *Time To First Token (TTFT)* when using (a) only OPT-1.3B, (b) only OPT-6.7B, and (c) HELIOS (*lower is better*). In (c), the vertical lines denote timestamps when a candidate re-assessment is initiated.

Figure 15 shows the improvements in number of requests served concurrently obtained from HELIOS. HELIOS has limited impact for smaller models. In contrast, for larger LLMs, where weights dominate GPU memory, such as Llama2-13B, HELIOS increases requests served per second by 6.05×.

Takeaway: With model sizes growing exponentially [110] and continued widespread adoption of LLMs, HELIOS unlocks a critical advantage, enabling us to service more requests per unit time with similar resource requirements.

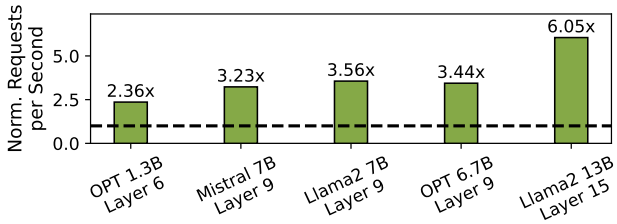


Figure 15. Normalized Requests per Second with HELIOS compared to standalone model without early-exits.

5.3 SLO: Response Time Optimization

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 14. TTFT is limited by the time taken to process all the tokens in the input prompt. 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× and 2.23× reduction in TTFT compared to OPT-1.3B and OPT-6.7B respectively. This is expected because of HELIOS's greedy model loading which services most tokens. With HELIOS, each token in the input prompt now traverses fewer layers, significantly reducing time spent in the input token processing phase. Also, HELIOS maximizes the total number of prompts serviced using early exits from both models combined. This is particularly evident for prompts 272 to 542 which corresponds to the CNN Daily mail dataset comprising long input prompts. HELIOS outperforms the OPT-6.7B model by up to 30× for some of these prompts because of the reduced number of layers traversed.

5.4 SLO: Accuracy Optimization

Next, we study the scenario when the user's goal is to maximize accuracy. Figure 16 shows the *perplexity (lower is better)* for three model selection cases. The perplexity of the larger model, OPT-6.7B, is lower (0.97×) than the smaller OPT-1.3B model. This is expected because larger models have many parameters and enhanced architecture enabling them to better capture the relationships between tokens. Specifically, for long context benchmarks, larger models are known [2, 111] to outperform smaller models. Hence, for the set of prompts corresponding to CNN Daily Mail [41], which comprises relatively long context inputs, HELIOS switches to using OPT-6.7B in real-time, marked as ❶ in Figure 16(c), to meet the target SLO of the user. On the other hand, HELIOS reverts back to OPT-1.3B later, marked as ❷ in Figure 16(c), because HELIOS evaluates that it offers accuracy comparable to OPT-6.7B in a more energy-efficient manner.

5.5 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 [112], emphasizing the impact of HELIOS. We also observe that the energy overheads associated with switching is minimal, comprising only 0.05× of the overall energy savings (10%) achieved.

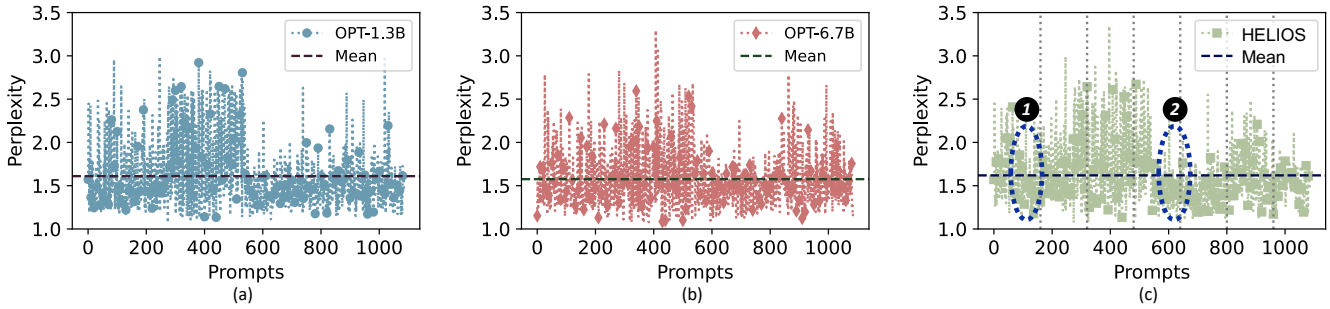


Figure 16. Comparison of *Perplexity (Accuracy)* when using (a) only OPT-1.3B, (b) only OPT-6.7B, and (c) HELIOS (*lower is better*). Vertical lines represent timesteps when a candidate re-assessment is initiated.

5.6 Understanding Dynamic Nature of HELIOS

Table 5 shows the distribution of the models used and exits taken by HELIOS for all input requests compared to statically selecting each model, under different SLOs. When the SLO is to maximize energy-efficiency, HELIOS steers more requests to take early exits by using a combination of both models in real-time. The entire model is used only in 7.9% and 0.6% of the cases in HELIOS, compared to 22.3% and 21.6% using OPT-1.3B and OPT-6.7B standalone respectively.

On the other hand, the OPT-6.7B model with 9 layers is better in terms of TTFT than more layers of the OPT-1.3B on an average. Thus, we see an increase in the number of prompts serviced by the OPT-6.7B model in the case where the SLO is to minimize response time. In contrast, when the SLO is to maximize accuracy, HELIOS steers more requests to the larger OPT-6.7B model, while maximizing early exits. About 34.6% of the prompts are now serviced using 9 layers of the OPT-6.7B model, a 2.26× increase compared to the case where the SLO is to maximize energy-efficiency.

Table 5. 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
SLO: Throughput Optimization						
HELIOS	70.19	1.38	6.78	20.90	0.14	0.61
SLO: Response Time Optimization						
HELIOS	49.3	0.20	0.90	48.9	0.10	0.6
SLO: Accuracy Optimization						
HELIOS	52.3	1.50	7.30	34.6	0.70	3.60
SLO: Energy Optimization						
HELIOS	74.3	1.80	7.90	15.3	0.10	0.6

5.7 Sensitivity Analysis

We study the impact of varying the hyper-parameters used in HELIOS. To report TTFT (response time), throughput, and energy-efficiency, we consider three different SLOs that optimize for the specific metric to enable a fair comparison. For example, to study the impact on TTFT, HELIOS assumes the SLO is to minimize response time.

5.7.1 Sensitivity to Re-assessment Interval (*RI*).

Increasing the *RI* has no impact for static model selection. With HELIOS, the throughput decreases at an *RI* of 200, because it encounters large prompts in that segment, where the smaller model is ineffective and therefore, needs to switch. With increasing *RI* the periodic candidate re-assessment now happens too infrequently, making it harder for HELIOS to adapt to the changing characteristics of the input prompts, as shown in Figure 17(a).

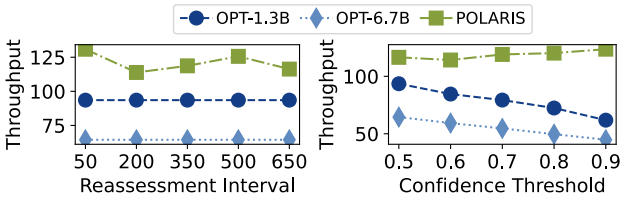


Figure 17. Impact of increasing *RI* and *TH* on throughput.

5.7.2 Sensitivity to Confidence Threshold (*TH*).

Increasing Confidence Threshold (*TH*) reduces the number of tokens taking early exits because it becomes harder to meet the exit criterion. Consequently, more tokens traverse more layers- reducing throughput. Figure 17(b) shows the throughput with increasing *TH*. We observe that HELIOS consistently outperforms the baseline- where either OPT-1.3B or OPT-6.7B is selected and used throughout.

6 Related Work

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

Comparison with Speculative Decoding: [15, 16, 113] uses two models, a main target model, and a smaller draft model. The draft model generates output tokens one-by-one, while the main model periodically verifies and corrects them in parallel. 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 [41] dataset. LayerSkip [14] incorporates early-exits into their speculative decoding framework for further benefits. However, this approach relies on task-specific fine-tuned models, limiting its generality. HELIOS can adaptively switch between such self-speculative models by monitoring their early-exit behavior in real-time, to further optimize inference efficiency based on the nature of prompts.

Hardware-Software Co-Design: BERT Loses Patience [49] proposes a technique for fast and robust inference where, if classifiers from the intermediate layers repeatedly predict the same token, then after a set threshold an early exit is taken. Edge-BERT [114] propose throttling the DVFS mechanism to meet the target latency by predicting future exit layers. HELIOS provides flexibility to optimize other key performance metrics, such as throughput and latency, beyond 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. As discussed earlier in Section 2, INFaaS [26] selects a model that meets SLOs of the task and performs inference with it, and Clipper [27] 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[94], and model parallelism, as used in AlpaServe [115] are complementary to HELIOS, and could be combined to scale our design to accommodate larger models on the GPUs. Similarly Splitwise [13], a technique which splits the prefill and the token generation phase across multiple GPUs complements the design of HELIOS.

7 Conclusion

Early-Exit LLMs (EE-LLMs) offer unique opportunities to balance the trade-offs between key performance metrics of inference serving. By using fewer layers to process trivial prompts and all layers otherwise, EE-LLMs improve throughput without compromising accuracy. However, the efficacy of EE-LLMs is limited by static model selection and prior knowledge about early exits, which heavily depends on the input task. We propose *HELIOS*, a software framework that dynamically selects a model and its early exit layers in real-time to adapt to the specific needs of the input task and efficiently serve incoming prompts. By greedily loading only a subset of layers of a selected model to serve most requests confidently, and by periodically reassessing a set of candidate models to adapt to the changing nature of input queries, HELIOS efficiently optimizes for the target performance metric while meeting all user specified SLOs. Our studies show that HELIOS achieves 1.48× throughput, 1.10× energy-efficiency, 1.39× lower response time, and 3.7× improvements in inference batch sizes compared to the baseline, when optimizing for the respective SLOs.

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. 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.

Appendix:

Algorithm 1 : Algorithm used in HELIOS

Input: Model Repository (MR), SLOs, Strategy
Output: Dynamic Model Selection
Parameters: **M** : Full Model; **M'** : Low Exit Model
CBC : Confidence Breach Counter; $\mathbf{CBC_{max}}$: Threshold

```
Candidates ← Top_k( MR (models that meet SLOs))
while prompts in requests do
    CBC, Serviced Prompts ← 0
    PHT[M,M'] ← Evaluate (Candidates)
    Chosen ← BestModel (PHT[M'], Strategy)
    while Serviced Prompts < Timeout do
        Serve (prompt, Selected)
        if confidence not met then
            CBC ← CBC + 1
            if CBC > CBC_max then
                if PHT[M(Selected)] < PHT[M'(Others)]
  then
                    // Load More Layers
                    Chosen ← M[Chosen]
                    CBC ← 0
                else     // Swap Model
                    Chosen ← M'[NextBestModel(PHT)]
                    CBC ← 0
                end if
            end if
        end if
    end while
end while
```

References

[1] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota, 2019.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[4] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[5] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

[6] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*, 2024.

[7] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024.

[8] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. *arXiv preprint arXiv:2405.04437*, 2024.

[9] Kostas Hatalis, Despina Christou, Joshua Myers, Steven Jones, Keith Lambert, Adam Amos-Binks, Zohreh Dannenhauer, and Dustin Dannenhauer. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 277–280, 2023.

[10] Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching. *arXiv preprint arXiv:2412.03594*, 2024.

[11] Jiayi Liu, Tinghan Yang, and Jennifer Neville. Cliqueparcel: An approach for batching llm prompts that jointly optimizes efficiency and faithfulness. *arXiv preprint arXiv:2402.14833*, 2024.

[12] Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*, 2023.

[13] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

[14] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[15] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36:39236–39256, 2023.

[16] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[17] Xiangjie Li, Chenfei Lou, Yuchi Chen, Zhengping Zhu, Yingtao Shen, Yehan Ma, and An Zou. Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8657–8665, 2023.

[18] Jingcun Wang, Bing Li, and Grace Li Zhang. Early-exit with class exclusion for efficient inference of neural networks. In *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pages 263–267. IEEE, 2024.

[19] Tianyi Shen, Chonghan Lee, and Vijaykrishnan Narayanan. Multi-exit vision transformer with custom fine-tuning for fine-grained image recognition. In *2023 IEEE International Conference on Image Processing (ICIP)*, pages 2830–2834. IEEE, 2023.

[20] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Multi-exit vision transformer for dynamic inference. *arXiv preprint arXiv:2106.15183*, 2021.

[21] Guanyu Xu, Jiawei Hao, Li Shen, Han Hu, Yong Luo, Hui Lin, and Jialie Shen. Lgvit: Dynamic early exiting for accelerating vision transformer. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9103–9114, 2023.

[22] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Single-layer vision transformers for more accurate early exits with less overhead. *Neural Networks*, 153:461–473, 2022.

[23] Ilias Leontiadis, Stefanos Laskaridis, Stylianos I Venieris, and Nicholas D Lane. It's always personal: Using early exits for efficient on-device cnn personalisation. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 15–21, 2021.

[24] Geonho Kim and Jongsun Park. Low cost early exit decision unit design for cnn accelerator. In *2020 International SoC Design Conference (ISOCC)*, pages 127–128. IEEE, 2020.

[25] Emanuele Lattanzi, Chiara Contoli, and Valerio Freschi. Do we need early exit networks in human activity recognition? *Engineering Applications of Artificial Intelligence*, 121:106035, 2023.

[26] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. {INFaaS}: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411, 2021.

[27] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A {Low-Latency} online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, 2017.

[28] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. {MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1049–1062, 2019.

[29] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 443–462, 2020.

[30] AWS. Amazon sagemaker, 2018. https://aws.amazon.com/sagemaker/.

[31] AWS. Amazon sagemaker neo, 2018. https://aws.amazon.com/sagemaker/neo/.

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

[33] Azure machine learning, 2018. https://docs.microsoft.com/en-us/azure/machine-learning/.

[34] Nvidia triton inference server„ 2020. https://github.com/triton-inference-server/server.

[35] Google cloud ai platform„ 2018. https://cloud.google.com/ai-platform/.

[36] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[37] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[38] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[39] Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 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.

[40] Xuchen Pan, Yanxi Chen, Yaliang Li, Bolin Ding, and Jingren Zhou. Ee-tuning: An economical yet scalable solution for tuning early-exit large language models. *arXiv preprint arXiv:2402.00518*, 2024.

[41] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Stefan Riezler and Yoav Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics.

[42] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

[43] Tal Schuster, Adam Fisch, Tommi Jaakkola, and Regina Barzilay. Consistent accelerated inference via confident adaptive transformers. *arXiv preprint arXiv:2104.08803*, 2021.

[44] Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. *arXiv preprint arXiv:2310.05424*, 2023.

[45] Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. Accelerating llama inference by enabling intermediate layer decoding via instruction tuning with lite. *arXiv e-prints*, pages arXiv–2310, 2023.

[46] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[47] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

[48] Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[49] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit, 2020.

[50] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

[51] Can large language models identify and correct their mistakes?, 2024. https://research.google/blog/can-large-language-models-identify-and-correct-their-mistakes/.

[52] Stephen Chung, Wenyu Du, and Jie Fu. Learning from failures in multi-attempt reinforcement learning, 2025.

[53] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

[54] Kv cache reuse, 2024. https://nvidia.github.io/TensorRT-LLM/advanced/kv-cache-reuse.html.

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

[56] Openai: Prompt cache, 2024. https://platform.openai.com/docs/guides/prompt-caching.

[57] Jiaqi Wang, Zhengliang Liu, Lin Zhao, Zihao Wu, Chong Ma, Sigang Yu, Haixing Dai, Qiushi Yang, Yiheng Liu, Songyao Zhang, et al. Review of large vision models and visual prompt engineering. *Meta-Radiology*, 1(3):100047, 2023.

[58] Louie Giray. Prompt engineering with chatgpt: a guide for academic writers. *Annals of biomedical engineering*, 51(12):2629–2633, 2023.

[59] Bertalan Meskó. Prompt engineering as an important emerging skill for medical professionals: tutorial. *Journal of medical Internet research*, 25:e50638, 2023.

[60] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

[61] Prompt engineering market size, share & trends analysis report by component (software, services), by technique (n-shot, generated knowledge), by application, by industry, by region, and segment forecasts, 2024 - 2030, 2024. https://www.grandviewresearch.com/industry-analysis/prompt-engineering-market-report/toc.

[62] Hugging face„ 2016. https://huggingface.co/.

[63] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[64] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[65] Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[66] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

[67] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[68] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

[69] Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. *arXiv preprint arXiv:2402.14992*, 2024.

[70] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery,

Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[71] Jonathan Pilault, Raymond Li, Sandeep Subramanian, and Christopher Pal. On extractive and abstractive neural document summarization with transformer language models. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 9308–9319, 2020.

[72] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.

[73] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[74] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*, 2022.

[75] Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Dylan Slack, Qin Lyu, et al. A careful examination of large language model performance on grade school arithmetic. *arXiv preprint arXiv:2405.00332*, 2024.

[76] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[77] Thorsten Brants, Ashok Popat, Peng Xu, Franz Josef Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, 2007.

[78] Zhiwei He, Tian Liang, Wenxiang Jiao, Zhuosheng Zhang, Yujiu Yang, Rui Wang, Zhaopeng Tu, Shuming Shi, and Xing Wang. Exploring human-like translation strategy with large language models. *Transactions of the Association for Computational Linguistics*, 12:229–246, 2024.

[79] Hanzhou Li, John T Moon, Saptarshi Purkayastha, Leo Anthony Celi, Hari Trivedi, and Judy W Gichoya. Ethics of large language models in medicine and medical research. *The Lancet Digital Health*, 5(6):e333–e335, 2023.

[80] Kai He, Rui Mao, Qika Lin, Yucheng Ruan, Xiang Lan, Mengling Feng, and Erik Cambria. A survey of large language models for healthcare: from data, technology, and applications to accountability and ethics. *arXiv preprint arXiv:2310.05694*, 2023.

[81] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[82] Rukmini Iyer, Mari Ostendorf, and Marie Meteer. Analyzing and predicting language model improvements. In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 254–261. IEEE, 1997.

[83] Dietrich Klakow and Jochen Peters. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38(1-2):19–28, 2002.

[84] Yang Gao, Wei Zhao, and Steffen Eger. Supert: Towards new frontiers in unsupervised evaluation metrics for multi-document summarization. *arXiv preprint arXiv:2005.03724*, 2020.

[85] Tingting He, Jinguang Chen, Liang Ma, Zhuoming Gui, Fang Li, Wei Shao, and Qian Wang. Rouge-c: A fully automated evaluation method for multi-document summarization. In *2008 IEEE International Conference on Granular Computing*, pages 269–274. IEEE, 2008.

[86] Oleg Vasilyev, Vedant Dharnidharka, and John Bohannon. Fill in the blanc: Human-free quality estimation of document summaries. *arXiv preprint arXiv:2002.09836*, 2020.

[87] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[88] AI@Meta. Llama 3 model card. 2024.

[89] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[90] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[91] Together Computer. Redpajama: an open dataset for training large language models, 2023.

[92] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020.

[93] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[94] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[95] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024.

[96] Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning, 2024.

[97] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023.

[98] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[99] Clémentine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, and Thomas Wolf. Open llm leaderboard v2. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard, 2024.

[100] Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard (2023-2024). https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard, 2023.

[101] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

[102] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning

benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664, 2021.

[103] Costas Mavromatis, Petros Karypis, and George Karypis. Pack of llms: Model fusion at test-time via perplexity optimization. *arXiv preprint arXiv:2404.11531*, 2024.

[104] Bhuvanashree Murugadoss, Christian Poelitz, Ian Drosos, Vu Le, Nick McKenna, Carina Suzana Negreanu, Chris Parnin, and Advait Sarkar. Evaluating the evaluator: Measuring llms' adherence to task evaluation instructions. *arXiv preprint arXiv:2408.08781*, 2024.

[105] Zachary Ankner, Cody Blakeney, Kartik Sreenivasan, Max Marion, Matthew L Leavitt, and Mansheej Paul. Perplexed by perplexity: Perplexity-based data pruning with small reference models. *arXiv preprint arXiv:2405.20541*, 2024.

[106] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[107] Ravi Ghadia, Avinash Kumar, Gaurav Jain, Prashant Nair, and Poulami Das. Dialogue without limits: Constant-sized kv caches for extended responses in llms. *arXiv preprint arXiv:2503.00979*, 2025.

[108] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

[109] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[110] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[111] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[112] Yuntao Wang, Yanghe Pan, Miao Yan, Zhou Su, and Tom H Luan. A survey on chatgpt: Ai-generated contents, challenges, and solutions. *IEEE Open Journal of the Computer Society*, 2023.

[113] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

[114] Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul N. Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. 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.

[115] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving, 2023.