SELA: Smart Edge LLM Agent to Optimize Response Trade-offs of AI Assistants

SHRESHTH TULI, Happening Technology, UK
GIULIANO CASALE, Imperial College London, UK
MANUEL ROVERI, Politecnico di Milano, Italy

The advent of Large Language Models (LLMs) has changed the way we process information today and has unlocked new ways of delivering intelligence to the user. One of the ways of interfacing with AI is via smart assistants and chatbots that take multimodal inputs. However, the diversity of input tasks imply the possibility of both latency-critical and complex input instructions for AI assistants. Further, LLMs cannot be deployed on the edge for low-latency outputs, as that presents challenges due to their high computational demands and memory requirements. This work explores such trade-offs and contributes a smart LLM selection policy, called SELA, that leverages a suite of LLM models with disparate characteristics to optimize overall quality of service (QoS). SELA uses a time-criticality and complexity predictor at the edge to identify the optimal LLM choice for a given input instruction. Experiments on public instruction benchmarks demonstrate that SELA provides 9% to 62% higher QoS scores compared to the state-of-the-art selection policies.

CCS Concepts: • **Computer systems organization → Cloud computing**; • **Information systems → Information systems applications**;

Additional Key Words and Phrases: LLMs, AI Assistants, Wearables, Cloud Computing.

ACM Reference Format:
Shreshth Tuli, Giuliano Casale, and Manuel Roveri. 2025. SELA: Smart Edge LLM Agent to Optimize Response Trade-offs of AI Assistants. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 9, 3, Article 130 (September 2025), 18 pages. https://doi.org/10.1145/3749483

1 Introduction

Large Language Models (LLMs) have profoundly transformed various domains of work, interaction, and functional execution, offering remarkable advancements in natural language processing and comprehension [9]. Substantial progress has been achieved in enhancing the accessibility of these potent tools, exemplified by models such as ChatGPT, Perplexity, and Bard. A notable recent development is Project Astra, which facilitates interaction between a cloud-based LLM and edge devices [3]. The shift towards edge-cloud computing has introduced a critical need to balance network latency and compute capacities efficiently. Edge devices offer the advantage of reduced latency and real-time processing capabilities, while cloud-based LLMs provide substantial computational power and advanced processing capabilities. Leveraging this trade-off is essential for optimizing performance and efficiency. The concerted efforts of diverse groups to develop LLMs tailored to specific tasks and input characteristics have resulted in a rich array of open source LLM models currently available. For example, Meta's Llama-2 models [33] are available in variants with 7 billion (7B), 13 billion (13B), and 70 billion (70B) parameters.

Authors' Contact Information: Shreshth Tuli, shreshth.tuli@happening.xyz, Happening Technology, London, UK; Giuliano Casale, g.casale@imperial.ac.uk, Imperial College London, London, UK; Manuel Roveri, Politecnico di Milano, Milan, Italy, manuel.roveri@polimi.it.

ACM 2474-9567/2025/9-ART130
https://doi.org/10.1145/3749483

As parameter count increases, there is a corresponding improvement in response quality. However, this also leads to an increase in the inference time, and consequently the response time of the model. This variety of advanced LLMs enables users and practitioners to choose the model that best fits their specific use-case requirements, balancing the benefits of edge and cloud computing to achieve optimal performance.

Challenges. Despite the advancements in LLM technology, deploying these models on edge devices presents significant scalability challenges. Ensuring the accessibility of LLMs remains a critical issue due to the high computational demands and prolonged turnaround times associated with cloud-based models. However, this presents significantly higher response times due to high network latency of cloud machines that are at multi-hop distance from the users [8; 21]. Running large LLMs directly on constrained battery powered edge devices is impractical because of their substantial computational and memory requirements [11]. On the other hand, using smaller models exclusively compromises performance, even when employing advanced distillation strategies and model quantization techniques, which still result in considerable memory footprints and accuracy losses [28; 42]. Additionally, when deploying LLMs as assistants in Internet of Things (IoT) devices such as smart glasses, the models must be generic and optimized for latency and response quality across a wide range of input instructions [19; 20]. This necessitates the dynamic selection of LLMs based on the specific input instruction to optimize the network latency and compute capacity trade-offs between edge and cloud device, a task that itself requires comprehension of the instruction. While this selection process could be managed by an LLM in the cloud, it would further increase the response time compared to time taken by the LLM to process the input.

Insights. Understanding the input instruction is essential to select the appropriate processing model to send the input as a prompt. However, this understanding becomes less useful if it is executed via an LLM in the cloud. To balance latency-critical and resource-intensive workload processing, we propose integrating the strengths of both edge and cloud processing. Specifically, some processing is performed at the edge, while the remaining tasks are offloaded to the cloud, thereby optimizing output quality and latency. At the edge, we identify which LLM to select, while the response generation occurs in the cloud. High-end models like GPT-4o or Gemini-Pro, though powerful, are cost-prohibitive and suffer from latency issues [12]. In contrast, smaller models like TinyLLaMA or SBert are more feasible for edge deployment but do not meet the performance requirements for mission-critical tasks [9; 27; 43]. Our research addresses this gap by proposing a dynamic approach to LLM deployment. The key insight is an edge model that predicts two factors: the complexity and time-criticality of the input instruction. Predicting complexity is relatively straightforward, whereas predicting time-criticality presents more challenges. Nonetheless, predicting these two quantifiable scores for an input instruction is far easier than generating the response itself, and we hypothesize that this can be achieved with reasonable accuracy at the edge. By leveraging these predictions, along with the known turnaround times and performance metrics of existing LLMs, we can make informed decisions about which model to deploy for a given task.

Contributions. We introduce an on-device prediction mechanism, referred to as SELA, designed to assess the complexity and time-criticality scores of each input instruction. This system is trained on a supervised dataset where a high-parameter model, such as GPT-4, serves as the judge, following the LLM-as-a-judge paradigm [44]. Additionally, our time-criticality predictor functions as an early exit network, leveraging the complexity score as a prior to expedite processing when feasible. By integrating these predictions with known processing times and performance metrics of various cloud-based LLMs, our approach selects the model that maximizes the quality of service (QoS) score. The QoS score is defined as a convex combination of output quality and latency, with weights proportional to the complexity and time-criticality scores. Further, one benefit of using edge-based model selection is that it can help selectively decide whether an input even needs to leave the device, thereby reducing unnecessary data transmission and potential research in the privacy space. Our approach utilizes a diverse array of LLMs, ranging from TinyLLaMA to Llama-2, to ensure optimal performance and efficiency. Our experimental results demonstrate that this model selection strategy, based on complexity and time-criticality predictions, surpasses traditional reinforcement learning baselines and other early exit networks in terms of QoS.

Specifically, SELA achieves QoS improvements ranging from 9% to 62% over other methods, showcasing its robust and efficient performance across various benchmarks. This innovative approach holds promise for enhancing the deployment of LLMs on edge devices, thereby making powerful language models more accessible and efficient.

2 Background and Related Work

Diversity in LLMs. In recent years, several LLMs have been proposed to provide a natural language response for a given input prompt. For instance, Meta released Llama-2 models, which come in different parameter configurations (7B, 13B, 70B) to cater to varying requirements of response quality and latency [33]. There are also variants of many LLMs that are specifically tuned for instruction-answering use-cases. One such example is Mistral 7B Instruct model [18], which is designed to excel in instructional tasks by generating clear and precise responses based on a prompt. This model is particularly effective for educational applications, where detailed and accurate explanations are crucial. Another such variant is Vicuna, which is a conversational LLM fine-tuned from LLaMA models, designed specifically for high-quality dialogue generation and interaction tasks [10]. It utilizes extensive instruction tuning and dialogue-specific datasets to enhance its ability to understand and respond to user inputs in a conversational context, making it ideal for applications such as customer support, virtual assistants, and interactive chatbots. Another model by Microsoft is Phi-2 [17]. It employs a hybrid architecture that combines deep learning with symbolic reasoning, enabling it to understand context and semantics better than traditional models. Specifically, Phi is tailored for use cases that require precise and contextually aware language processing, such as legal document analysis and medical diagnostics, where the stakes for accuracy are exceptionally high. However, its computational demands and specialized design make it less suitable for real-time or resource-constrained environments, highlighting the need for more adaptable solutions like our proposed SELA. Rasgulla-7b [1] has been developed specifically for conversational AI applications in multilingual environments. It leverages a 7-billion parameter architecture optimized for handling diverse languages and dialects, providing robust performance in customer service and real-time translation scenarios.

However, the sheer variety of available models poses a challenge in dynamically selecting the most appropriate model for a given input, especially in scenarios that require real-time processing on edge devices. The current methods primarily rely on static selection, failing to adapt to the dynamic nature of input instructions. This requires a more flexible approach, such as our proposed method, which integrates on-device prediction mechanisms to dynamically assess and select the optimal model based on input complexity and time-criticality.

Efficient LLM Inference. There are also lightweight variants of LLaMA, such as TinyLLama, designed for efficient deployment on resource-constrained edge devices [43]. It achieves this by utilizing model compression techniques such as pruning and quantization to reduce the computational and memory requirements while maintaining adequate performance for less complex tasks, making it ideal for applications requiring quick and efficient processing without the need for high computational power or extensive memory, such as real-time language translation and basic conversational agents. Similarly, TinyMistral [18] is designed for efficient LLM inference on constrained devices by employing aggressive model pruning and quantization techniques. This model is specifically tailored for use cases requiring real-time natural language processing on edge devices with limited computational resources, such as smart home assistants and wearable technology. However, TinyMistral's aggressive optimizations often lead to significant performance trade-offs, making it less suitable for tasks demanding high accuracy or complex understanding. This highlights the need for our proposed method, which aims to dynamically balance resource utilization and performance by predicting input complexity and time-criticality, ensuring optimal model deployment for diverse tasks in an edge-cloud environment.

Another model in this category is Palmyra [36], which is designed to facilitate cross-lingual transfer, particularly targeting low-resource languages by leveraging high-resource language models. It operates by training on a multilingual corpus and utilizing transfer learning techniques to adapt high-resource language knowledge

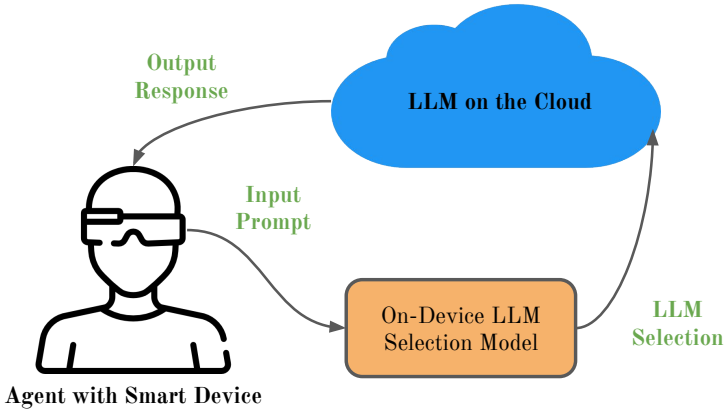

Fig. 1. System Model demonstrating the presence of a smart-device as an interface for a human user that sends an input instruction as a natural language prompt. The model selects appropriate LLM, which is run on the cloud, and returns response to the user to maximize Quality of Service.

to low-resource contexts. This model is particularly useful for applications in multilingual settings where the availability of training data for certain languages is limited, making it ideal for tasks such as translation, language generation, and understanding in underrepresented languages. Despite its specialization, Palmyra's focus on low-resource language adaptation does not address the broader challenges of real-time model selection and efficiency optimization on edge devices, necessitating our proposed hybrid edge-cloud approach.

Model Selection. Recent advancements have explored the use of RL to dynamically select among multiple large language models (LLMs) based on input characteristics. PickLLM introduces a context-aware RL-assisted framework that routes queries to the most suitable LLM by optimizing a reward function balancing cost, latency, and accuracy [30]. The system employs stateless Q-learning or gradient ascent to adaptively choose the optimal model for each query, demonstrating improvements over static selection methods. Other methods use policy-gradient networks [40] to identify the optimal LLM model as a model action. Similarly, ConsistentEE formulates the early exiting process as a reinforcement learning problem. A policy network decides whether an instance should exit or continue processing, aiming to maintain prediction consistency and reduce computational overhead [41].

Early-exit methods can also be trained for the task of selecting the appropriate LLM for given input. BranchyNet, a technique introduced for early exit networks, involves creating multiple exit points within a deep neural network, allowing predictions to be made at different stages based on the complexity of the input [32]. This method aims to reduce inference time by exiting early for simpler inputs and can be used for LLM selection. Another early-exit method is Zero-Time-Waste (ZTW) which is designed to minimize wasted computation time by dynamically selecting the appropriate model based on the predicted complexity of the input [37]. ZTW attempts to balance computational efficiency with accuracy, but it often struggles with accurately predicting the complexity of diverse inputs, leading to suboptimal model selection and performance. Further, such methods are unaware of the downstream task of LLM selection and hence, unlike SELA, do not leverage complexity or time-criticality information that LLMs can provide.

Table 1. Summary of the main notation

Symbol	Meaning
x	Input Instruction in natural language text
M_i	the i-th LLM that can process x
$f(x;\theta)$	Model selection for input x
$\alpha(x)$	Complexity Score for input x
$\beta(x)$	Time-Criticality Score for input x
$\mu(x, M_i(x))$	Response Quality Score of model M_i for input x
$\phi(x, M_i)$	Latency Score of model M_i for input x
$Q(x, M_i)$	Quality of Service Score of model M_i for input x

3 Methodology

3.1 System Model and Problem Formulation

In this work, we target a standard edge-cloud environment where we have a smart IoT device as an assistant, such as smart glasses or smartphones. Tasks are natural language input instructions provided as a prompt to the IoT device. The system model is shown in Figure 1 that illustrates the interaction between a human user equipped with a smart device and a cloud-based LLM with a summary of the presented notation shown in Table 1. The process begins with the user issuing an input instruction as a natural language prompt via the smart device. This input is then processed by an on-device LLM selection model, which determines the most suitable LLM to handle the prompt. The selected LLM, hosted on the cloud, processes the prompt and generates an appropriate output response. This response is then delivered back to the smart device of the user, ensuring an optimal quality of service. Specifically, for a given input instruction x, the goal is to select an appropriate LLM from a set of available models $\{M_1, M_2, M_3, \ldots, M_n\}$, execute x on the selected model (say M_i), and return the result. We represent the selection model by $f(x;\theta)$ such that

$$M_i = f(x;\theta), \tag{1}$$

where θ are the parameters of the selector model. Each input prompt x has an associated ground truth **complexity score** $\alpha(x)$ and **time-criticality score** $\beta(x)$, which are determined by a high-parameter LLM such as GPT-4o and are scalars in the range $[0, 1]$. However, LLMs such as GPT-4o are prone to have biases and hallucinate, say criticality or complexity of tasks. To mitigate this, we provide example-driven prompting, where we provide examples of tasks that are too complex and easy or too time-critical or not so critical. A task such as "give me some game suggestions" would have time-criticality score of 0 while "how do I treat my wasp sting immediately" would have a time-criticality score of 1. Further, a task such as "tell me how to heat milk" would have a complexity score of 0 while "tell me how to design an aircraft" would have a complexity score of 1. We provide GPT-4o with these examples and the input prompt to generate $\alpha(x)$ and $\beta(x)$ for each x.

Now, for each execution of the LLM model, we generate a score in the range $[0, 1]$ for the output $M_i(x)$ via a high-parameter model, such as GPT-4o. Here, we provide the input prompt x and the response $M_i(x)$ and ask GPT-4o to give a **response quality score** denoted by $\mu(x, M_i(x))$. GPT-4o is asked to consider the correctness and usefulness of the response $M_i(x)$ given input prompt x. Further, we also consider an inverted normalized **latency score**:

$$\phi(x, M_i) = 1 - \frac{t(x, M_i) - \min(t(x, M_i) \forall M_i)}{\max(t(x, M_i) \forall M_i) - \min(t(x, M_i) \forall M_i)}, \tag{2}$$

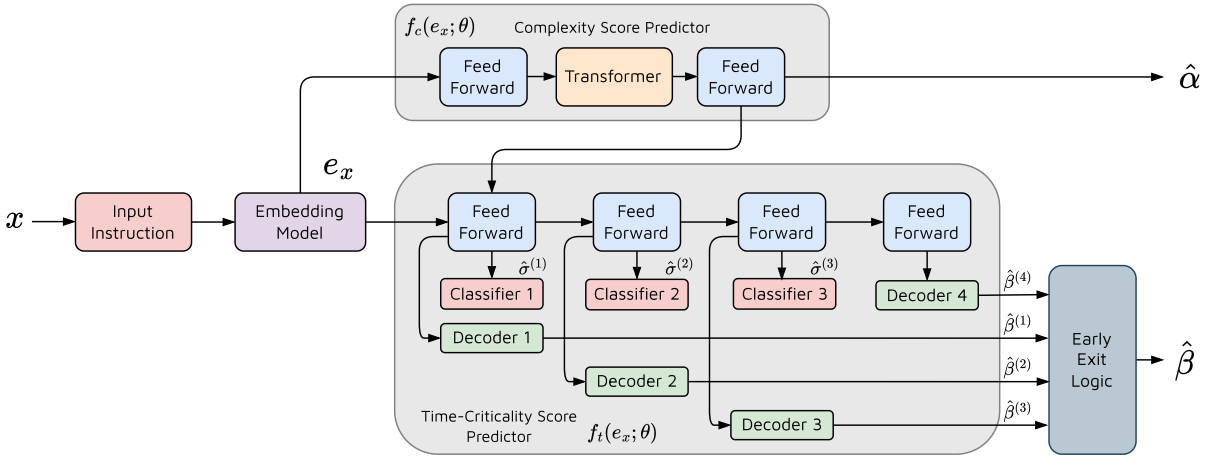

Fig. 2. System Model demonstrating the presence of a smart-device as an interface for a human user that sends an input instruction as a natural language prompt. The model selects appropriate LLM, which is run on the cloud, and returns response to the user to maximize quality of service.

where $t(x, M_i)$ is the time to execute LLM M_i for input prompt x, the fraction in the above equation is min-max scaled response time for the input, and we have this fraction subtracted from one to keep this score positive and a metric to maximize.

The QoS score for the output $M_i(x)$ generated by the selected LLM is given as

$$Q(x, M_i) = \alpha(x) \cdot \mu(x, M_i(x)) + \beta(x) \cdot \phi(x, M_i), \tag{3}$$

i.e., the QoS score is a linear combination of the response quality and latency scores where the weights are the complexity and time-criticality scores for the input prompt x, respectively.

Thus, the problem is formulated as follows. We need to identify the optimal selection model $f(\cdot;\theta)$, such that the above score is maximized for a diverse range of input prompts.

3.2 SELA

To address the problem of selecting the appropriate LLM based on the input instruction x, we propose a supervised learning approach using an early exit neural network to predict the complexity score $\alpha(x)$ and the time-criticality score $\beta(x)$ for each input instruction. We then use this predictor to identify the optimal LLM for each input instruction. Steps detailed below.

Data Collection. We first use publicly available instruction datasets such as MT Bench [44] and HH Bench [5] (more details in Section 4). For training, we use a uniform-sampler to get a fraction of these datasets and denote this training set by $X = \{x_1, x_2, x_3, \ldots, x_m\}$. For each input instruction, we prompt GPT-4o to provide the complexity and time-criticality scores $\alpha(x)$ and $\beta(x)$ as described in Section 3.1. We then use the dataset $\{(x_i, \alpha(x_i), \beta(x_i))\}_{i=1}^{m}$ as the training dataset.

Supervised Score Predictor. We now train a supervised model that predicts the complexity and time-criticality scores for each input instruction x and represents the outputs as

$$\hat{\alpha}_i, \hat{\beta}_i = f(x_i;\theta), \tag{4}$$

where $\hat{\alpha}_i$ and $\hat{\beta}_i$ are predicted complexity and time-criticality scores, respectively. *Insight 1:* We hypothesize that we do not need an LLM with all its foundational priors to predict these scores for a given input. Instead, we can

use text embeddings from simpler models such as Sentence-Bert [27] or Jina-Bert [13]. The latter is an augmented version of Bert that uses attention with linear biases to make it suit better for long input instructions [26]. Text embeddings from such models are dense feature vectors for natural-language input instruction text x. Such representations capture the semantic meaning of sentences effectively and are rich enough to provide meaningful input to a supervised model for predicting complexity and time-criticality. To give an intuitive example, predicting these scores can be done simply by extracting if certain words such as "urgently", "emergency" for time-criticality and "explain", "give steps" for complexity, could give strong indication on what these scores should be; thus demonstrating that predicting such scores is significantly easier compared to generating a response. The major advantage of this is that this enables us to embed this selector model within the edge device.

Model Architecture. Having dense feature representations for an input x, we now design an early-exit neural network to predict $\hat{\alpha}_i$ and $\hat{\beta}_i$ for input x_i (see Figure 2). Early exit models are designed to provide exit points at different layers of the network, allowing the model to stop processing once a confident prediction is made [22]. This is particularly useful for time-criticality prediction, as it ensures that the system can quickly respond to urgent inputs without unnecessary computation. As described above, we first obtain a dense feature representation of x as e_x. The selection of this embedding model depends on the memory capacity of the edge device (see section 4 for details on the embedding model used in experiments). *Insight 2:* We decouple the problem of predicting the two scores, first predicting the complexity score and then using an early-exit network with the complexity score as an input to predict time-criticality score. This is because the complexity of an input instruction is often tied to measurable factors such as the length of the input, the syntactic and semantic intricacy, and the need for advanced language understanding or generation capabilities. These factors are generally more straightforward to quantify and analyze using established natural language processing (NLP) techniques [39]. Further, complexity prediction relies more on the intrinsic properties of the input text rather than external factors. This intrinsic focus makes it less context-dependent compared to time-criticality, which can be influenced by external and situational factors. *Insight 3:* Allowing complexity prediction to be used as a guiding prior for time-criticality prediction facilitates in mitigating challenges such as dependence on external factors such as the current state of the user, the environment, or the specific application requirements. These factors can be dynamic and fluctuate in real-time, adding complexity to the prediction process and requiring long-horizon multi-step reasoning. Finally, unlike time-criticality, there exist well-defined metrics and features in NLP to assess the complexity of text, such as syntactic tree depth, number of entities, sentiment analysis, and lexical diversity [6; 23]. These features can be extracted from the input and used to train models that predict complexity with relatively high accuracy.

Thus, we first pass the input embedding e_x to a complexity score predictor denoted as $f_c(e_x;\theta)$. We use a transformer neural network [35] to implement this. The self-attention mechanism in transformers allows the model to weigh the importance of different parts of the input instruction dynamically. This is particularly useful for understanding the context in natural language, which is crucial for accurately assessing the complexity of an instruction. Further, transformers excel at capturing long-range dependencies and contextual relationships within the input text. Complexity in language often arises from such dependencies, where the meaning and difficulty of an instruction can depend on the interplay between distant words or phrases. Transformers are well-suited to handle these dependencies effectively. Thus, we represent the complexity predictor as

$$\begin{aligned} e_x^1 &= \text{FeedForward}(e_x), \\ e_x^2 &= \text{Transformer}(e_x^1), \\ \hat{\alpha} &= \text{FeedForward}(e_x^2), \end{aligned} \tag{5}$$

or, in short $\hat{\alpha} = f_c(e_x;\theta)$. Here, feed-forward layers help reduce the size of the input and consequently the memory footprint of the model. Time-criticality often depends on a few strong local features (e.g., presence of urgency

words) and does not require modeling long-range dependencies. Also, time-criticality prediction benefits from a lightweight and fast architecture (especially for early exit), and feedforward networks are more computationally efficient and quicker to evaluate than Transformers. Now that we have the predicted complexity score $\hat{\alpha}$, we forward it to a time-criticality score predictor, represented as $f_t(e_x, \hat{\alpha}; \theta)$. This is implemented as a feed-forward early-exit model. Feedforward networks are computationally efficient and can provide quick predictions, which is essential for time-critical applications. For the first exit, we have two outputs, a binary class output $\hat{\sigma}^{(1)}$ and a score prediction $\hat{\beta}^{(1)}$. The former is a prediction of whether to exit at this stage and the latter is the output of this exit. These are calculated as

$$\begin{aligned} b^{(1)} &= \text{FeedForward}(e_x, \hat{\alpha}), \\ \sigma^{(1)} &= \text{sigmoid}(\text{FeedForward}(b^{(1)})), \\ \hat{\beta}^{(1)} &= \text{FeedForward}(b^{(1)}). \end{aligned} \tag{6}$$

Assuming, we have n exits, all downstream early exits are

$$\begin{aligned} b^{(i)} &= \text{FeedForward}(b^{(i-1)}), \\ \sigma^{(i)} &= \text{sigmoid}(\text{FeedForward}(b^{(i)})), \\ \hat{\beta}^{(i)} &= \text{FeedForward}(b^{(i)}), \end{aligned} \tag{7}$$

where $i \in [2, n]$ for the feed-forward layers and decoder, but $i \in [2, n-1]$ for the classifier as that is not needed in the final layer. Figure 2 shows an architecture for $n = 4$ exits. The prediction of the binary output $\sigma^{(i)}$ allows us to exit early; thus, we give the earliest output $\hat{\beta}^{(k)}$ such that the class prediction is positive and all previous exit class predictions are negative. Specifically,

$$\hat{\beta} = \hat{\beta}^{(k)} \;\Big|\; \left(\hat{\sigma}^{(k)} > 0.5\right) \wedge \left(\hat{\sigma}^{(i)} \leq 0.5 \;\forall\; i < k\right), \tag{8}$$

or, in short $\hat{\beta} = f_t(e_x, \hat{\alpha}; \theta)$. This above equation is represented as the "early exit logic" in Figure 2.

Model Training. To train the neural network in a supervised fashion, we define the first loss function as the mean squared error (MSE) between the predicted scores and the ground truth scores,

$$\mathcal{L}_{MSE} = \frac{1}{m} \sum_{i=1}^{m} \left[(\hat{\alpha}_i - \alpha_i)^2 + (\hat{\beta}_i - \beta_i)^2 \right]. \tag{9}$$

This loss function ensures that the score predictions are accurate. To train the early-exit classifiers, we use a discretization threshold for the prediction error, represented as τ. When the absolute prediction error for an exit is below this threshold, then the ground-truth class is 1 (network can exit at i), else 0. Specifically,

$$\sigma^{(i)} = \mathbb{1}\left[|\hat{\beta}^{(i)} - \beta^{(i)}| \leq \tau \right]. \tag{10}$$

By setting an error threshold, we ensure that the model only exits early if the predictions are sufficiently accurate. This prevents premature exits that could lead to significant errors in the final predictions. The error threshold τ acts as a hyperparameter that can be tuned based on the specific application requirements. For applications prioritizing speed, a higher threshold can be set, while for those requiring higher accuracy, a lower threshold can be used. The loss function for training the exit classifiers can be formulated using binary cross-entropy, which is suitable for binary classification tasks. For a given input x, let $\hat{\sigma}^{(i)}$ be the probability that the classifier at exit i decides to exit (i.e., predicts class 1), and let $\sigma^{(i)}$ be the ground truth class as defined above. The binary cross-entropy loss for the exit classifier at stage i is given by

$$\mathcal{L}^{(i)} = -\left[\sigma^{(i)} \log(\hat{\sigma}^{(i)}) + (1 - \sigma^{(i)}) \log(1 - \hat{\sigma}^{(i)}) \right]. \tag{11}$$

The overall loss for the network, considering all exit classifiers, is the sum of the losses at each exit stage

$$\mathcal{L}_{BCE} = \sum_{i=1}^{n-1} \mathcal{L}^{(i)}, \tag{12}$$

where n is the total number of exit stages in the network. By minimizing this loss function during training, the network learns to make accurate decisions about when to exit early based on the prediction error, guided by the predefined error threshold. This approach ensures a balance between maintaining high-quality predictions and achieving efficient resource utilization. The overall loss to train the model becomes

$$\mathcal{L} = \mathcal{L}_{MSE} + \lambda \cdot \mathcal{L}_{BCE}, \tag{13}$$

where λ is a hyperparameter that controls the emphasis placed on the early-exit decisions relative to the accuracy of the predicted complexity and time-criticality scores. This linear combination allows balancing the trade-off between the accuracy of the predicted scores and the effectiveness of the early-exit mechanism.

LLM Selection. To implement an effective LLM selection policy, we can leverage heuristics based on the confidence bounds of the score and execution time for each LLM, derived from the training set data. Specifically, we use the lower confidence bound (LCB) for the response quality and the latecy scores. This approach ensures a conservative estimate of the expected quality and a pessimistic estimate of the expected latency, thus guiding the selection towards models that are likely to perform reliably under various conditions. Thus, the LCB of response quality score becomes

$$LCB(\mu, M_i) = \underset{x \in X}{\text{Mean}}(\mu(x, M_i(x))) - z \cdot \frac{\text{SD}_{x \in X}(\mu(x, M_i(x)))}{\sqrt{m}}, \tag{14}$$

and similarly, the LCB of the latency score becomes

$$LCB(\phi, M_i) = \underset{x \in X}{\text{Mean}}(\phi(x, M_i) - z \cdot \frac{\text{SD}_{x \in X}(\phi(x, M_i))}{\sqrt{m}}. \tag{15}$$

Here, z is the z-score corresponding to the desired confidence level (e.g., 1.96 for 95% confidence), SD represents the standard-deviation, and m is the number of samples in the training data.

Now, for any given input x, we predict the complexity and time-criticality scores as $\hat{\alpha}, \hat{\beta} = f(x, \theta)$ and then we select the LLM model that has the highest expected QoS

$$M* = \arg\max_{M_i} \left[\hat{\alpha} \cdot LCB(\mu, M_i) + \hat{\beta} \cdot LCB(\phi, M_i) \right]. \tag{16}$$

This selection policy allows us to make informed decisions about which LLM to deploy for a given task by considering both the quality and latency in a conservative manner. By using the lower confidence bound for scores and the upper confidence bound for execution times, we account for variability and uncertainty in model performance, ensuring robust and efficient LLM deployment.

4 Experiments

We compare SELA with several state-of-the-art model selection methods in reinforcement learning, such as DDPG [15], PPO [29], DQN [24; 30] and SAC [14; 41]. We also compare SELA with other early-exit network based model selectors, such as BranchyNet [32] and Zero-Time-Waste [37] (see Section 2 for more details). We implement all neural networks as feed-forward instead of convolution networks as run inference on dense vector embeddings instead of images. We use hyperparameters of the baseline models as presented in their respective papers.

We train all deep learning models using the PyTorch-1.8.0 [25] library. All model training and experiments are performed on a system with configuration: M3 Max CPU, 128GB RAM and Mac OS Sonoma. In the experiments,

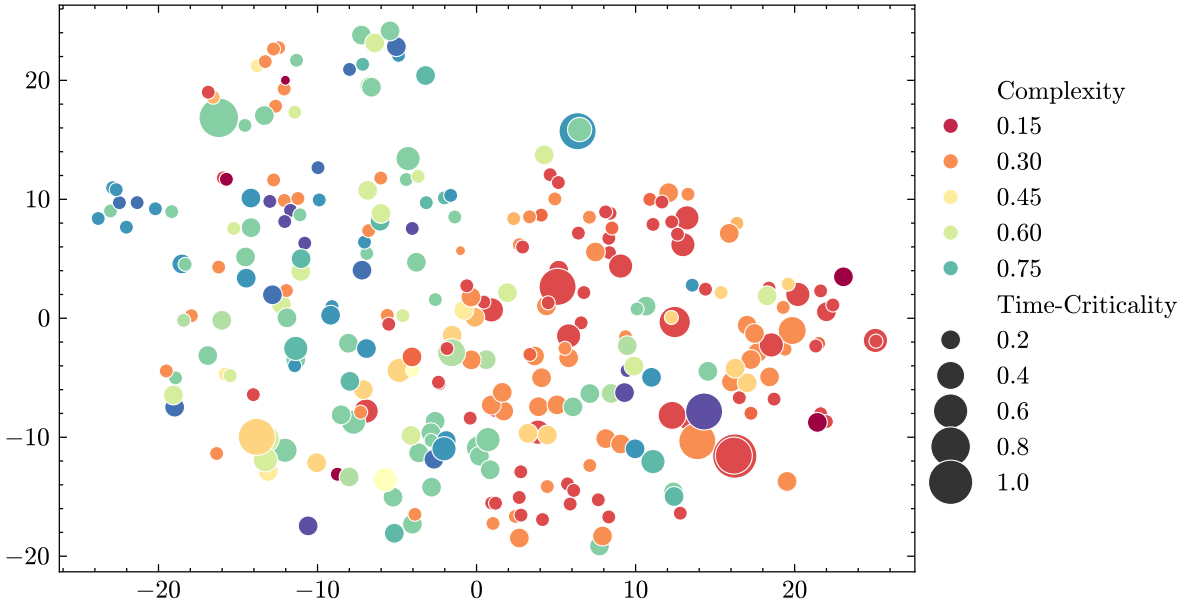

Fig. 3. t-SNE Plot demonstrating the distribution of the input instruction embeddings in the test set. The plot shows that the embedding model is able to differentiate between the input task complexity. Tasks with higher complexity are to the left (ex: "tell me how to set up this IKEA desk"), whereas easier tasks are shown on the right (ex: "what is the height of Eiffel Tower").

we consider the network latency of the edge device where SELA runs to be 10ms whereas that of the LLM models running in the cloud to be 200ms (typical for LLMs hosted on Huggingface) [7; 34; 38].

4.1 Evaluation Setup

Selecting the Embedding Model. We consider a typical smart glasses such as Quest 3 and Apple Vision Pro. For a typical smart glasses application, we can use the TAPAS model [16], which has a memory usage of ~8.62MB with FLOAT16 precision. TAPAS is a BERT-like model that focuses weak supervision consisting of denotations instead of logical forms, reducing its memory footprint. This embedding model can easily fit in the memory capacity of typical smart glasses. While running small models such as TAPAS on edge devices reduces latency and preserves user privacy, it does incur non-trivial power consumption, which can be problematic for battery-constrained devices like smart glasses and mobile wearables. In our current design, we justify this trade-off by prioritizing low-latency responsiveness and selective data privacy for tasks where these factors are critical. However, for scenarios where energy conservation is paramount, or when user tasks are less time-sensitive, fully offloading inference to cloud-based LLMs, coupled with optimization techniques such as prompt caching, model quantization, or low-bit transmission, may provide a more sustainable alternative. A promising future extension of SELA would involve dynamic mode switching, where the system can favor edge processing or cloud processing based on current battery levels, user preferences, or network quality, thereby achieving a more adaptive balance between energy efficiency and service quality.

To visualize high-dimensional embeddings by the TAPAS model we generate a t-SNE (t-distributed Stochastic Neighbor Embedding) plot where we project them into a two-dimensional space, preserving local structure and grouping similar data points together to highlight clusters and patterns, often aiding in understanding complex

datasets. The t-SNE plot in Figure 3 demonstrates that a simple model like TAPAS-tiny is sufficiently effective for generating dense vector embeddings that distinguish between input task complexity and time-criticality. This conclusion is drawn from the clear separation observed in the plot, where tasks with higher complexity are predominantly located on the left side (e.g., "tell me how to set up this IKEA desk"), while easier tasks are concentrated on the right side (e.g., "what is the height of Eiffel Tower"). Additionally, the varying sizes of the points, which indicate time-criticality, are well-distributed across the plot, showing that the model captures this dimension effectively. Despite being lightweight, TAPAS-tiny can produce embeddings that maintain the semantic and functional integrity required to differentiate tasks along these two axes. This efficiency makes it an optimal choice for edge deployments where computational resources are limited, proving that even smaller models can achieve robust performance in distinguishing task characteristics without the need for more complex and resource-intensive embedding models.

In our evaluation, the latency for each input instruction is measured by recording timestamps using OS-level system functions (specifically, Python's `time.time()` API). The measurement starts just before the instruction is sent to the model (upload start) and ends immediately after the full output response is received (download complete). Thus, our recorded latency includes both the inference time of the model as well as the network transmission time between the smart device and the cloud server. For edge models, the network latency is assumed to be negligible (10ms) due to local execution, while for cloud-based models, we assume an average network latency of 200ms based on prior measurements [22; 34]. However, we acknowledge that real-world network conditions are highly variable, but having fixed latency assumptions allows us to perform controlled experiment where the experiments are sensitized for LLM selection choice and not network volatility. In the future, we could account for network volatility effects by adding network variance as part of time-criticality predictor and consider upper confidence bounds for risk-averse decision making.

Portfolio of LLM Models. For our experiments, we evaluated the proposed LLM selection policies using a diverse set of language models, ensuring comprehensive assessment through varying parameter sizes, architectures, and training objectives. The models include phi-1_5, phi-2, and phi-2-super, TinyMistral-248M-v2.5, TinyLlama-1.1B-Chat-v1.0, palmyra-small, Mistral-7B-Instruct-v0.2, palmyra-3B, vicuna-13b-v1.5, vicuna-7b-v1.5, RasGulla1-7b (see Section 2 for more details). These models range from compact, resource-efficient options like TinyMistral and TinyLlama, to larger, high-performance models such as vicuna-13b and phi-2-super. This selection spans various architectures and optimization targets, from chat-focused designs to instruction-based tasks. By including models optimized for both efficiency and high performance, we ensure our experiments cover a wide spectrum of performance and resource usage scenarios. Consequently, this diverse model set is adequate for robustly testing the proposed LLM selector and baseline methods, eliminating the need for additional models.

Hyperparameter Tuning. Hyperparameters were tuned using the Optuna package, where 50 trials were used [4] with Tree-structured Parzen Estimator (TPE) sampler, which is well-suited for this task due to its efficiency and effectiveness in navigating complex hyperparameter spaces. The hyperparameter selection gave $n = 4$ exits, $z = 1.44$ (85% confidence) and $\tau = 0.5$. To implement the SELA early-exit neural network, we use LeakyReLU activation function.

Metrics. We consider the time to select the LLM as a metric, the time to execute an LLM model on the cloud for the input instructions, response quality scores and QoS as defined in eq. 3. The response time in QoS calculation is the sum of selection and execution times for each input instruction.

4.2 Datasets

We use four publicly available instruction datasets. **MT Bench** is a widely used benchmark that evaluates the performance of language models in multilingual tasks [44]. It includes a variety of tasks such as translation, cross-lingual understanding, and multilingual question answering. This benchmark assesses the ability of the

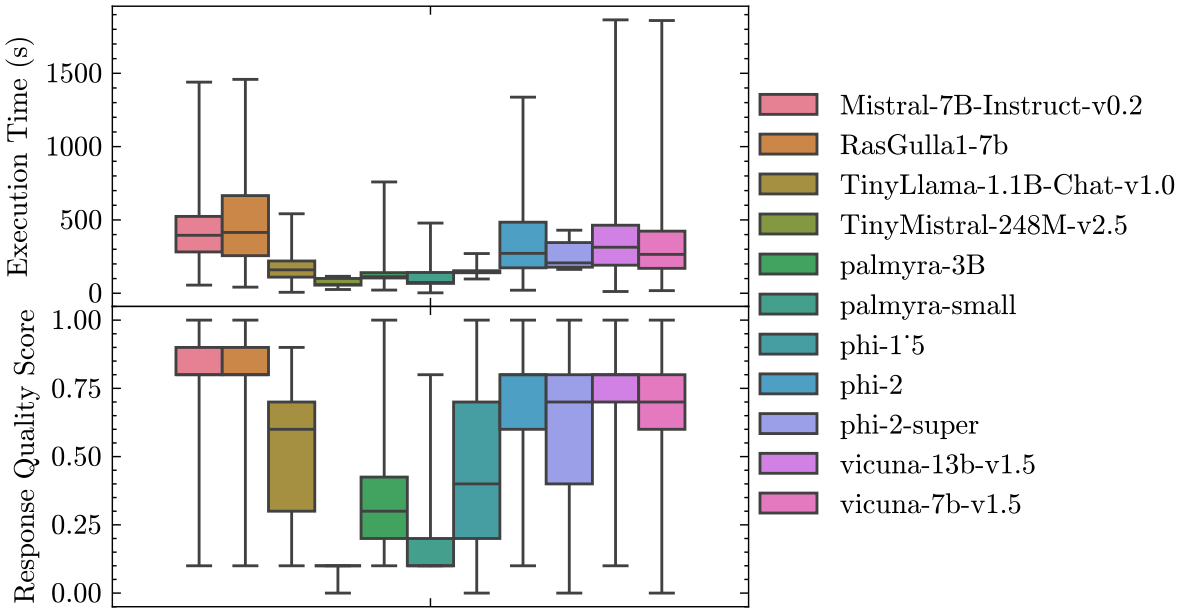

Fig. 4. Distribution of response quality scores and latencies of all LLMs considered in the evaluation setup. For larger models, such as `Mistral-7B-Instruct-v0.2` or `vicuna-13b-v1.5`, we see that the response quality score is higher than the smaller models such as `palmyra-small` or `TinyMistral-248M-v2.5`. However, larger models also tend to have higher execution time compared to the smaller models.

model to handle diverse languages and its performance in multilingual settings. **Vicuna Bench** focuses on evaluating models in natural language understanding and generation tasks within the context of scientific literature and technical documentation. The benchmark includes tasks such as summarization, question answering, and information extraction from technical texts [44]. This benchmark measures the model capability to process and generate accurate information in specialized domains, highlighting its utility in technical and scientific applications. **IHAP Bench** consists of a rich set of human-generated prompts designed to simulate realistic assistant interactions [2]. It includes detailed instructions and queries that a human assistant might encounter, making it a valuable resource for testing the model ability to understand and respond to intricate and nuanced instructions. **HH Bench** is designed to evaluate the helpfulness and harmlessness of language model outputs [5]. This dataset includes prompts that require the model to provide helpful and safe responses, avoiding any harmful or inappropriate content. It is particularly useful for assessing the ethical and practical aspects of model deployment in real-world scenarios.

We then use the GPT-4o model to identify complexity (α) and time-criticality (β) scores for each instruction prompt. We also use GPT-4o as a judge model to identify the response quality score for response from each LLM model in our evaluation setup.

4.3 Analysing LLM Performance and SELA Outputs

Figure 4 gives the distribution of the response quality scores and latency for each LLM. Notably, high-end models like Mistral-7B-Instruct-v0.2 and vicuna-13b-v1.5 exhibit high response quality scores, but also show significant execution times, reflecting their computational intensity and resource demands. Conversely, smaller models such

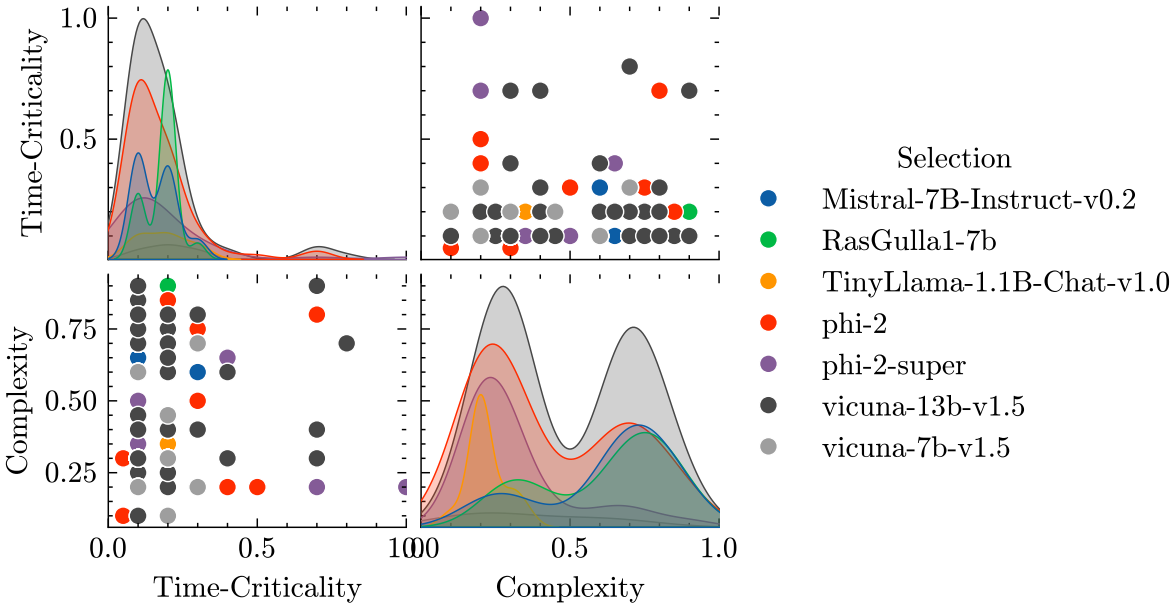

Fig. 5. Visualization of LLM selections by SELA for the test set. It demonstrates that for less complex instructions, SELA prefers `phi-2` or `phi-2-super`, but for highly complex instructions, it uses `RasGulla-7b` or `Mistral-7B-Instruct-v0.2`. The model does never selects `palmyra-3B`, `palmyra-small`, `phi-1.5`, `TinyMistral-248M-v2.5`.

as TinyMistral-248M-v2.5 and TinyLlama-1.1B-Chat-v1.0, though faster in execution, display lower response quality scores, indicative of their trade-off between speed and performance. This distribution validates our selection policy, demonstrating it effectively leverages the strengths of various models based on task complexity and time-criticality. The clear delineation in performance and latency among the models underscores the importance of dynamic model selection in optimizing both response quality and execution efficiency, affirming the rationale behind the proposed SELA framework.

Figure 5 illustrates the selection distribution of LLM models by SELA across the test set, highlighting its ability to intelligently choose appropriate models based on predicted complexity and time-criticality scores. The scatter plots and density distributions reveal that for less complex instructions, SELA predominantly selects the phi-2 model, as indicated by the concentration of red points in the lower complexity range. This choice reflects an optimal balance between performance and resource utilization, given the efficiency and adequacy of phi-2 for simpler tasks (see Figure 4). Conversely, for highly complex instructions, SELA frequently opts for RasGulla-7b, evidenced by the green points clustered in the higher complexity range (see Figure 4). This selection underscores the model capacity to allocate more powerful LLMs where greater computational resources and nuanced understanding are necessary. The distribution also shows that time-criticality influences model selection; for instructions with higher time-criticality, SELA often prefers models such TinyLlama-1.1B-Chat-v1.0 and phi-2, which can provide faster responses. Overall, the intelligent selection by SELA, guided by predicted complexity and time-criticality scores, demonstrates its ability to maximize quality-of-service by dynamically adapting to the specific demands of each input instruction. This adaptive approach ensures efficient resource utilization while maintaining high response quality, validating the effectiveness of SELA model selection policy.

Table 2. Performance comparison of SELA with baseline methods on the four datasets. Sel. Time: selection time, Exec. Time: execution time, Score: response quality score (μ), QoS: quality of service score (Q). The best scores are highlighted in bold.

Method	HH Bench				IHAP Bench			
	Sel. Time	Exec. Time	Score	QoS	Sel. Time	Exec. Time	Score	QoS
Random	**0.000±0.000**	332.089±11.367	5.950±0.300	0.291±0.023	**0.000±0.000**	229.722±9.123	6.400±0.287	0.325±0.023
DDPG	0.037±0.003	**287.562±9.842**	6.300±0.262	0.307±0.022	0.036±0.002	236.142±7.553	7.700±0.103	0.386±0.020
PPO	0.039±0.004	287.564±9.841	6.300±0.262	0.307±0.022	0.037±0.002	236.143±7.553	7.700±0.103	0.386±0.020
DQN	0.035±0.003	369.259±11.064	7.200±0.147	0.370±0.023	0.035±0.002	**229.040±6.932**	7.000±0.215	0.352±0.023
SAC	0.040±0.004	498.444±12.061	7.550±0.201	0.418±0.039	0.039±0.004	304.999±9.797	7.150±0.241	0.413±0.023
Branchy	0.044±0.004	370.805±8.089	7.300±0.200	0.389±0.029	0.046±0.003	348.028±12.826	7.400±0.157	0.445±0.029
ZTW	0.047±0.004	298.534±6.655	5.850±0.296	0.289±0.021	0.048±0.005	279.406±11.423	7.500±0.157	0.410±0.025
SELA w E1	0.054±0.007	423.236±11.794	7.150±0.198	0.418±0.031	0.059±0.008	272.889±7.482	7.250±0.215	0.420±0.024
SELA w/o EE	0.053±0.005	378.788±11.051	6.800±0.244	0.369±0.023	0.051±0.004	358.847±11.618	7.900±0.085	0.466±0.023
SELA	0.053±0.004	433.107±12.676	**7.800±0.181**	**0.423±0.024**	0.053±0.005	323.222±10.097	**7.980±0.176**	**0.496±0.027**
Method	**MT Bench**				**Vicuna Bench**			
	Sel. Time	Exec. Time	Score	QoS	Sel. Time	Exec. Time	Score	QoS
Random	**0.000±0.000**	**309.629±18.204**	5.312±0.324	0.485±0.041	**0.000±0.000**	**184.587±8.083**	4.812±0.310	0.310±0.029
DDPG	0.040±0.008	471.118±15.956	6.188±0.302	0.648±0.032	0.035±0.003	292.259±9.926	6.625±0.200	0.463±0.019
PPO	0.043±0.009	471.122±15.956	6.188±0.302	0.648±0.032	0.040±0.003	292.264±9.926	6.625±0.200	0.463±0.019
DQN	0.040±0.009	524.235±19.806	6.375±0.294	0.685±0.037	0.037±0.002	215.806±5.557	6.062±0.198	0.398±0.020
SAC	0.045±0.009	525.749±21.349	6.125±0.316	0.653±0.044	0.039±0.004	285.604±9.711	6.562±0.219	0.454±0.022
Branchy	0.050±0.010	406.509±17.581	6.875±0.253	0.692±0.040	0.046±0.003	255.545±6.736	6.938±0.195	0.479±0.021
ZTW	0.053±0.012	475.745±19.770	6.750±0.249	0.681±0.036	0.051±0.005	244.230±8.495	6.938±0.184	0.455±0.019
SELA w E1	0.046±0.016	481.150±14.122	6.375±0.320	0.676±0.040	0.060±0.006	261.234±7.710	7.062±0.173	0.483±0.022
SELA w/o EE	0.058±0.011	574.881±12.826	7.062±0.284	0.749±0.054	0.054±0.003	298.772±8.223	7.375±0.109	0.516±0.018
SELA	0.059±0.013	566.069±17.610	**7.375±0.260**	**0.790±0.035**	0.054±0.004	269.800±6.141	**7.562±0.188**	**0.535±0.019**

Figure 6 provides a visualization of the early-exit mechanisms within the SELA model, highlighting the relationship between exit points and the predicted complexity and time-criticality of input instructions. The plot illustrates that as we progress from early exits (Exit 1) to later exits (Exit 5), the complexity of the input instructions (green bars) steadily increases, while the time-criticality (blue bars) decreases. This pattern demonstrates the ability of SELA to accurately identify and respond to time-critical instructions by opting for earlier exits, thus minimizing inference time and enhancing responsiveness. Conversely, for more complex tasks, SELA allows for later exits, providing the necessary computational depth to ensure high-quality responses. The selection time (orange bars), which is directly proportional to inference time, also increases with later exits, ensuring optimal resource utilization and response quality by tailoring the computational effort to the specific demands of each input instruction.

4.4 Comparison with Baselines

The performance of SELA was compared with several baseline methods across four benchmark datasets: HH Bench, IHAP Bench, MT Bench, and Vicuna Bench. We also include a random selection policy as a baseline for completeness. Table 2 summarizes the results, detailing the selection time, execution time, response quality score, and quality of service (QoS).

In the HH Bench dataset, SELA achieve a QoS score of 0.423, significantly outperforming all baseline methods, with SAC and BranchyNet being the closest competitors at 0.389 and 0.413, respectively. Notably, the response quality score for SELA is 7.800, which surpasses the other models, indicating its superior ability to balance quality and efficiency. For the IHAP Bench dataset, SELA again demonstrates the highest QoS score of 0.496. The next best-performing model was BranchyNet with a QoS score of 0.445, followed by SAC at 0.410. The execution time

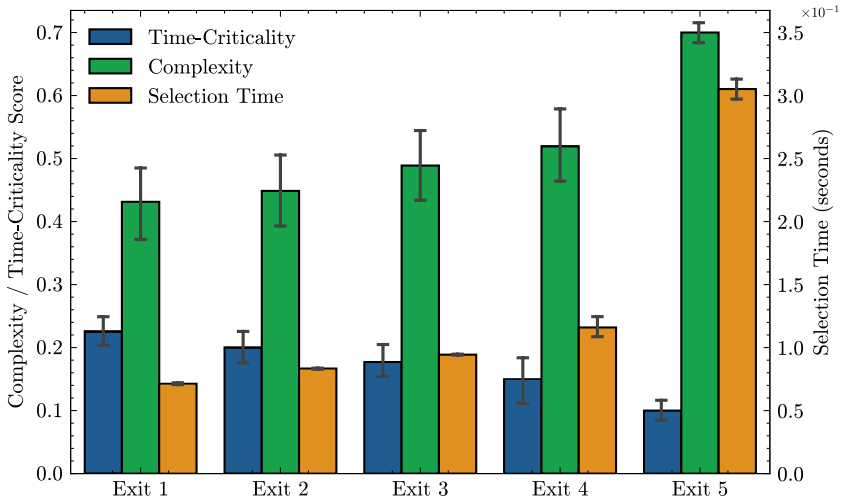

Fig. 6. Visualization of the early-exits in the model. As we go from early to late exits, the complexity of the input instruction increases whereas the time-criticality decreases. This demonstrates that the model is able to identify that for time-critical instructions it must exit early. Similarly, the selection time (viz. directly proportional to the inference time) increases as SELA takes exits later.

and response quality score for SELA were 323.222 seconds and 7.980, respectively, highlighting its capability to deliver high-quality responses efficiently. On the MT Bench dataset, SELA achieved a notable QoS score of 0.790, outperforming baselines such as BranchyNet (0.692) and SAC (0.653). The execution time for SELA was 566.069 seconds, and the response quality score was 7.375 further demonstrating its balanced approach to quality and latency. In the Vicuna Bench dataset, SELA achieves the highest QoS score of 0.535, with a response quality score of 7.562. The closest baseline, BranchyNet, has a QoS score of 0.479, demonstrating the superior performance of SELA in optimizing for both response quality and time-critical requirements.

4.5 Ablation Analysis

To understand the impact of the early-exit strategy in SELA, we conduct an ablation analysis by comparing the full SELA model with two variations: SELA w E1, which exits at the first exit, and SELA w/o EE, which always exits at the last exit of the neural network.

In the HH Bench dataset, SELA w E1 have a QoS score of 0.418, while SELA w/o EE scores 0.369, both significantly lower than the full SELA model score of 0.423. This indicates that while early exits can reduce execution time, they may compromise response quality, and the flexibility of the dynamic exit strategy in SELA is crucial for optimal performance. Similarly, in the IHAP Bench dataset, SELA w E1 and SELA w/o EE scored 0.420 and 0.466, respectively, compared to 0.496 for SELA. The full model ability to dynamically choose exit points based on task complexity and time-criticality ensures a higher QoS. In the MT Bench dataset, SELA w E1 has a QoS of 0.676, and SELA w/o EE scores 0.749±0.054, whereas SELA achieves 0.790. This further underscores the importance of the early-exit mechanism in balancing efficiency and performance. For the Vicuna Bench dataset, the QoS scores for SELA w E1 and SELA w/o EE were 0.483 and 0.516, respectively, both lower than SELA which has 0.535. The superior performance of the full model highlights the benefits of its adaptive exit strategy.

Overall, the ablation analysis confirms that the dynamic early-exit strategy in SELA is essential for optimizing both response quality and execution efficiency, significantly outperforming fixed exit strategies.

5 Conclusions

In this paper, we introduced SELA, a novel framework for optimizing the selection of LLMs based on input complexity and time-criticality scores. Our approach leverages a diverse array of LLMs, ranging from lightweight models suitable for edge deployment to more complex models for handling intricate tasks. The results demonstrate that SELA consistently outperforms traditional reinforcement learning-based selectors and static early-exit strategies, achieving significant improvements in QoS across various benchmarks. The dynamic exit mechanism in SELA ensures efficient utilization of resources while maintaining high response quality, intelligently adapting to the specific demands of each input instruction.

For future work, we plan to explore several extensions of this research. First, we will investigate the integration of real-time feedback mechanisms to continuously update the selection policy based on user interactions and evolving task requirements. Furthermore, expanding the scope of SELA to include multi-modal inputs, such as visual and auditory data, could further broaden its applicability and robustness in diverse real-world IoT scenarios and also mitigate the bias or hallucination effects in ground-truth labeling via models such as GPT-4o using quality checks [31] or fine-tuning on human-annotated data. Finally, we will examine the potential of federated learning approaches with variable network conditions to enable local adaptation to changing network behaviors and improve model selection policies. This can be achieved by simulating varying conditions and validate SELA's adaptability in more realistic environments, while preserving data privacy and security, enabling wider adoption of SELA in volatile and privacy-sensitive applications.

6 Acknowledgments

This work was carried out in the EssilorLuxottica Smart Eyewear Lab, a Joint Research Center between EssilorLuxottica and Politecnico di Milano. Giuliano Casale, from Imperial College London, contributed to the paper independently via Imperial Consultants.

References

[1] AbacusResearch/RasGulla1-7b · Hugging Face — huggingface.co. https://huggingface.co/AbacusResearch/RasGulla1-7b. [Accessed 14-07-2024].

[2] Dahoas/instruct-human-assistant-prompt · Datasets at Hugging Face — huggingface.co. https://huggingface.co/datasets/Dahoas/instruct-human-assistant-prompt. [Accessed 15-07-2024].

[3] Project Astra — deepmind.google. https://deepmind.google/technologies/gemini/project-astra/. [Accessed 14-07-2024].

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[5] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[6] M. Ballesteros, B. Bohnet, S. Mille, and L. Wanner. Data-driven deep-syntactic dependency parsing. *Natural Language Engineering*, 22(6):939–974, 2016.

[7] F. Bang. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 212–218, 2023.

[8] K. Cao, S. Hu, Y. Shi, A. W. Colombo, S. Karnouskos, and X. Li. A survey on edge and edge-cloud computing assisted cyber-physical systems. *IEEE Transactions on Industrial Informatics*, 17(11):7806–7819, 2021.

[9] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

[10] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

[11] Q. Dong, X. Chen, and M. Satyanarayanan. Creating edge AI from cloud-based llms. In *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*, pages 8–13, 2024.

[12] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

[13] M. Günther, J. Ong, I. Mohr, A. Abdessalem, T. Abel, M. K. Akram, S. Guzman, G. Mastrapas, S. Sturua, B. Wang, et al. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents. *arXiv preprint arXiv:2310.19923*, 2023.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[15] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

[16] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online, July 2020. Association for Computational Linguistics.

[17] M. Javaheripi, S. Bubeck, M. Abdin, J. Aneja, S. Bubeck, C. C. T. Mendes, W. Chen, A. Del Giorno, R. Eldan, S. Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 1(3):3, 2023.

[18] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[19] M. Kazemitabaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, and T. Grossman. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2024.

[20] A. Khurana, H. Subramonyam, and P. K. Chilana. Why and when llm-based assistants can go wrong: Investigating the effectiveness of prompt-based interactions for software help-seeking. In *International Conference on Intelligent User Interfaces*, pages 288–303, 2024.

[21] J. Li, D. Li, Y. Ye, and X. Lu. Efficient multi-tenant virtual machine allocation in cloud data centers. *Tsinghua Science and Technology*, 20(1):81–89, 2015.

[22] Y. Matsubara, M. Levorato, and F. Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–30, 2022.

[23] W. Medhat, A. Hassan, and H. Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[26] O. Press, N. A. Smith, and M. Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

[27] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[28] B. Rokh, A. Azarpeyvand, and A. Khanteymoori. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology*, 14(6):1–50, 2023.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] D. Sikeridis, D. Ramdass, and P. Pareek. Pickllm: Context-aware rl-assisted large language model routing. *arXiv preprint arXiv:2412.12170*, 2024.

[31] G. Sriramanan, S. Bharti, V. S. Sadasivan, S. Saha, P. Kattakinda, and S. Feizi. Llm-check: Investigating detection of hallucinations in large language models. *Advances in Neural Information Processing Systems*, 37:34188–34216, 2024.

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

[33] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[34] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings. COSCO: Container orchestration using co-simulation and gradient based optimization for fog computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):101–116, 2021.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[36] A. Weaver. Palmyra LLMs empower secure, enterprise-grade generative AI for business — writer.com. https://writer.com/blog/palmyra/. [Accessed 14-07-2024].

[37] M. Wołczyk, B. Wójcik, K. Bałazy, I. T. Podolak, J. Tabor, M. Śmieja, and T. Trzcinski. Zero time waste: Recycling predictions in early exit neural networks. *Advances in Neural Information Processing Systems*, 34:2516–2528, 2021.

[38] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[39] J. Yang, J. Redi, G. Demartini, and A. Bozzon. Modeling task complexity in crowdsourcing. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 4, pages 249–258, 2016.
[40] D. Ye, Y. Lin, Y. Huang, and M. Sun. Tr-bert: Dynamic token reduction for accelerating bert inference. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5798–5809, 2021.
[41] Z. Zeng, Y. Hong, H. Dai, H. Zhuang, and C. Chen. Consistentee: A consistent and hardness-guided early exiting method for accelerating language models inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19506–19514, 2024.
[42] L. Zhang, Y. Shi, Z. Shi, K. Ma, and C. Bao. Task-oriented feature distillation. *Advances in Neural Information Processing Systems*, 33:14759–14771, 2020.
[43] P. Zhang, G. Zeng, T. Wang, and W. Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.
[44] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009