RT-LM: Uncertainty-Aware Resource Management for Real-Time Inference of Language Models

Yufei Li[1] Zexin Li[1] Wei Yang[2] Cong Liu[1]
[1]University of California, Riverside [2]University of Texas at Dallas
[1]{yli927,zli536,congl}@ucr.edu [2]wei.yang@utdallas.edu

***Abstract*—Recent advancements in language models (LMs) have gained substantial attentions on their capability to generate human-like responses. Though exhibiting a promising future for various applications such as conversation AI, these LMs face deployment challenges on various devices due to their extreme computational cost and unpredictable inference latency. Such varied inference latency, identified as a consequence of uncertainty intrinsic to the nature of language, can lead to computational inefficiency and degrade the overall performance of LMs, especially under high-traffic workloads. Unfortunately, the bandwidth of these uncertainty sources is extensive, complicating the prediction of latency and the effects emanating from such uncertainties. To understand and mitigate the impact of uncertainty on real-time response-demanding systems, we take the first step to comprehend, quantify and optimize these uncertainty-induced latency performance variations in LMs. Specifically, we present RT-LM, an uncertainty-aware resource management ecosystem for real-time inference of LMs. RT-LM innovatively quantifies how specific input uncertainties, recognized within the NLP community, adversely affect latency, often leading to an increased output length. Exploiting these insights, we devise a lightweight yet effective method to dynamically correlate input text uncertainties with output length at runtime. Utilizing this quantification as a latency heuristic, we integrate the uncertainty information into a system-level scheduler which explores several uncertainty-induced optimization opportunities, including uncertainty-aware prioritization, dynamic consolidation, and strategic CPU offloading. Quantitative experiments across five state-of-the-art LMs on two hardware platforms demonstrates that RT-LM can significantly reduce the average response time and improve throughput while incurring a rather small runtime overhead.**

***Index Terms*—Language model, uncertainty, real-time system**

I. Introduction

The recent surge in the development and dissemination of language models (LMs) such as ChatGPT has significantly reshaped the landscape of natural language processing (NLP) [1]–[4]. This advancement holds immense promise for a multitude of applications, including multi-lingual robots and voice control devices integral to the future of smart homes [5]–[7]. Despite the impressive capability to generate human-like responses, these state-of-the-art LMs present a formidable challenge when attempting to deploy them on various devices due to their complex computational behaviors and unpredictable real-time inference capabilities [8], [9]. With the increasing demand for real-time language processing, server-backed systems, such as online chatbots (e.g., ChatGPT manages over 10 million daily queries) and live-translation services, exemplify the need for devices that can efficiently process simultaneous requests from multiple users, especially during peak times.

A set of recent works seek to enhance the inference latency of on-device LMs by crafting an array of model optimization techniques, including quantization [10], pruning [11], [12], and distillation [13]. These techniques aim at decreasing model complexity (thus the computational demand) while preserving their accuracy. Nonetheless, a knowledge gap persists in understanding and exploring the correlation between an input text and the corresponding inference latency within a given LM from a system-level perspective.

The NLP community has recently brought to light various sources of uncertainties [14]–[18], which have been shown to negatively impact model's accuracy and may introduce significant variations in the lengths of generated responses. Take, for example, a broad and ambiguous question such as "*Can you tell me the history of art?*". This could prompt a LM to generate lengthier outputs, given that the history of art spans millennia and includes a multitude of cultures, styles, periods, and artistic movements. Intuitively, the longer output a LM generates, the greater the inference latency, as each output token is sequentially generated with negligible computational difference [19], [20]. These sources of uncertainties, often intrinsic to the nature of language understanding and generation, can stem from varying data distributions [21], [22], intricate model architectures [23], or even the non-deterministic parallel computing behaviors at runtime [24], rendering the induced latency more complex and challenging to manage. Consequently, it is critical to understand and mitigate such uncertainties due to their potential to induce non-trivial inference latency and computational inefficiency, or even hinder the prompt delivery of dialogue generation (DG) due to degraded system performance.

This work is specifically motivated by the following queries: (i) What is the intrinsic correlation between an input text's uncertainty characteristics and the subsequent computational demand (and thus, the inference latency) for a given LM, such as why two syntactically similar inputs may necessitate dramatically different inference latencies? (ii) Is it feasible to devise a lightweight approach to predict an input's computational demand at runtime? and (iii) Can the system-level resource manager exploit these quantified input characteristics to improve latency performance during inference? Understanding the quantifiable correlation between an input text and its computational demand is critical, as it could unveil novel

opportunities for system-level optimization, thereby enhancing the performance and efficiency of LMs deployed on embedded devices, e.g., by deferring the execution of inputs with high computational demand thus reducing head-of-line blocking.

Our research attempts to comprehend, quantify, and optimize these uncertainty-induced variations on latency performance in LMs. We propose a cohesive ecosystem that integrates an application-level uncertainty quantification framework with a system-level uncertainty-aware resource manager. The application-level framework aims to precisely quantify task uncertainties and their potential impacts on latency. Simultaneously, the system-level resource manager utilizes the provided estimations to make informed decisions on resource allocation and task scheduling, thereby mitigating the detrimental effects of uncertainties on system performance.

Contributions. In this paper, we propose an uncertainty-ware resource management ecosystem, namely `RT-LM`, for real-time on-device LMs. Specifically, `RT-LM` features three technical novelties: 1) It first quantitatively reveals how major input uncertainties—well-defined by the NLP community—negatively impact latency. Our findings demonstrate that uncertainty characteristics of an input text may notably increase the output length, i.e., the number of tokens in the generated response; 2) Building on this insight, we develop a lightweight yet effective method that can quickly correlate and quantify the output length for an input text at runtime, considering a comprehensive set of uncertainties defined by the NLP community; 3) Leveraging this quantification as a heuristic of latency, we incorporate the uncertainty information of each input into system-level scheduler that performs several optimizations, including uncertainty-aware prioritization, dynamic consolidation, and strategic utilization of CPU cores.

We implement `RT-LM` mainly on an edge server. We evaluate the response time and throughput across five state-of-the-art LMs[1], namely DialoGPT [25], GODEL [26], BlenderBot [27], BART [28], and T5 [29]. We utilize `RT-LM` four widely-researched benchmark datasets: *Blended Skill Talk* [30], *PersonaChat* [31], *ConvAI2* [32], and *Empathetic Dialogues* [33]. For both the models and datasets, we use the versions released by Hugging Face.

Evaluation results demonstrate that `RT-LM` achieves:

- **Efficiency**: `RT-LM` outperforms all compared methods by a significant margin in most cases, improving the maximum response time by up to 30% and throughput by up to 40% compared to uncertainty-oblivious baselines.
- **Efficacy across a range of behaviors**: The tested workloads include five LMs with diverse task uncertainty characteristics and varied workload settings.
- **Robustness under malicious scenarios**: `RT-LM` is resilient when facing adversarial conditions, effectively mitigating the impact of malicious tasks by resource management.
- **Runtime overhead**: The design and implementation of `RT-LM` is efficient, incurring a rather small runtime latency and memory usage.

[1]While there are larger models like ChatGPT that offer impressive capabilities, their resource-intensive nature makes them less viable for deployment.

TABLE I: Types of linguistic uncertainty, their definitions and example statements or questions.

Type	Definition	Statement/Question
Structural ambiguity	Uncertainty related to multiple possible parse structures, leading to outputs with varying lengths.	"John saw a boy in the park with a telescope."
Syntactic ambiguity	Uncertainty arising from multiple part-of-speech tags of a word, resulting in different interpretations.	"Rice flies like sand."
Semantic ambiguity	Uncertainty stemming from words with multiple meanings, leading to varying interpretations.	"What's the best way to deal with bats?"
Vague expressions	Uncertainty arising from broad concepts or highly-generalized topics that demand specific analysis.	"Tell me about the history of art."
Open-endedness	Questions or statements that lack a single definitive answer and require providing relevant context, background, and explanations.	"What are the causes and consequences of poverty in developing countries?"
Multi-partness	Questions or statements containing multiple sub-questions or topics, which demand detailed answers.	"How do cats and dogs differ in behavior, diet, and social interaction?"

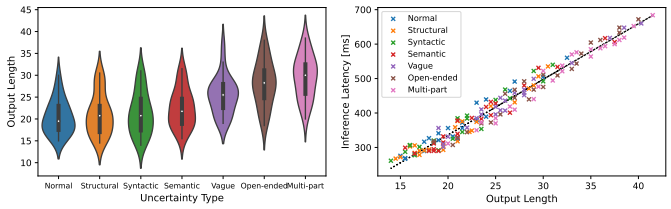
II. Background and Challenges

A. Dialogue Generation using LMs

Recently, pre-trained LMs such as ChatGPT and GPT-4 [1] have emerged as a dominant force in the field of dialogue generation (DG). These models are characterized by their large size and are often trained on vast amounts of textual data, which demonstrate remarkable capabilities in understanding and generating human-like responses across a wide range of tasks. A key property of these models is the *autoregressive* generation process [9], where output tokens are generated sequentially with each new token being conditioned on the previously generated tokens. Consequently, the output length plays a pivotal role in determining the inference latency of a LM, as generating longer sequences inherently requires more time. Depending on the nature of inputs, a LM may generate outputs of varied lengths. For instance, a query that has clear and concise meanings may elicit a brief response, whereas an ambiguous or broad query may demand a considerably longer output. This variability, often called *linguistic uncertainty* [34] by the NLP community, in output length and the subsequent impact on latency, can pose significant challenges when deploying LMs on resource-constrained devices, as the performance requirements and computational constraints must accommodate a wide range of potential latencies.

B. Sources and Impacts of Linguistic Uncertainty

Linguistic uncertainty is a challenging and diverse subdomain in NLP, which often leads to multiple interpretations of inputs and potentially varied outputs in dialogue systems. The language and linguistics community has well-defined a categorization of linguistic uncertainty that encompasses the majority of uncertainty sources, including three types of



(a) Uncertainty distribution for each type (b) Output length v.s. inference latency

Fig. 1: Observations of (a) distribution of LM output lengths for inputs with different uncertainty types, and (b) the correlation between LM output lengths and inference latency.

lexical ambiguity (structural ambiguity [14], [35], syntactic ambiguity [15], [36], semantic ambiguity [37], [38]), vague expressions [16], open-ended questions [17], [39], and multi-part questions [18] that demand comprehensive answers and additional explanations. Their definitions and example statements or questions are listed in Table I.

III. Key Observations and Ideas

A. Uncertainty-Induced Negative Impact on LM Latency and the Root Cause

We conducted a comprehensive set of studies investigating the correlation between inputs' uncertainty characteristics and the resulting inference latency of several LMs. Specifically, we create 1,000 utterances for each of the six uncertainty types (defined in Sec. II-B) and record the averaged output length as well as inference latency across DialoGPT, GODEL, BlenderBot, BART, and T5, as shown in Fig. 1a. We observe that all types of linguistic uncertainties lead to longer outputs and non-trivially larger latencies to varying degrees. Specifically, vague expressions, open-endedness, and multi-partness are generally more deterministic compared to the three types of lexical ambiguities. This can be attributed to that modern neural networks (NNs) lack uncertainty awareness and are prone to overconfidence when making decisions [40], which results in LMs understanding one potential interpretation and respond accordingly without seeking further clarifications. Furthermore, semantic ambiguity has a more significant impact on output lengths than structural and syntactic ambiguities. We speculate that this is because some words with multiple meanings such as *"trunk"* or *"monitor"* are more likely to cause confusions for a LM and thereby triggering longer responses, e.g., by enumerating all potential interpretations of a word sense and asking for explanations.

Fig. 1b plots the correlation between inference latency and output length for sentences that contain different types of uncertainties. We observe that inference latency is proportional to the output length, with longer outputs generally requiring larger inference latencies. Some sentences with uncertainties such as open-endedness and multi-partness may even take over 700ms for a LM to generate corresponding responses, which is 2∼4 times the latency of normal sentences. This presents a substantial opportunity for system-level optimization, as resource manager can leverage this uncertainty impact as an estimation of task execution times to enhance system efficiency and resource utilization.

Listing 1: Code for measuring vague expression scores.

```
def vague_expressions_score(sentence, weight=1):
    vague_count = 0
    words = word_tokenize(sentence)
    stem_words = [lemmatizer.lemmatize(word) for word in
        words]
    # VAGUE_WORDS are pre-defined in the literature
    for phrase in VAGUE_WORDS:
        matches =
            stem_words.count(lemmatizer.lemmatize(phrase))
        vague_count += matches
    return weight * vague_count
```

B. Predicting the Output Length for a Given Input

Upon observing that inference latency is determined by output length, we develop methods that can accurately yet efficiently predict such length for a given input at runtime. As discussed earlier, uncertainty of an input text may increase the output length and thus negatively impact inference latency. Our methods shall take uncertainty into account when making such predictions.

> **Uncertainty score**: In this work, we define uncertainty score for an input text as the estimated number of tokens (output length) required to formulate a comprehensive and unambiguous response that sufficiently addresses the posed inquiry.

Input length. Intuitively, longer inputs may lead to LMs generating longer outputs, even without considering uncertainty. We demonstrate the impact of this naive heuristic on output lengths in Fig. 2a. We observe that although the correlation is not deterministic and noisy, longer input lengths generally induce longer generated outputs. This inspires us to further improve it by considering uncertainty.

Single rule. We measure the intensity of each uncertainty using hand-crafted rules introduced in the literature. Specially, we use the spaCy[2] language tool to tokenize input text and obtain the Part-of-Speech (PoS) tag for each token in the original text. Then, we quantify uncertainty scores by searching for pre-defined patterns inherently existing in each uncertainty source using regular expressions. Listing 1 shows an example code for quantifying vague expression uncertainty. Note that for input sentences that do not contain the defined six uncertainty sources, we use input lengths as their single rule scores. We evaluate the correlation between single rule scores and the output lengths for inputs containing the corresponding type of uncertainty in Fig. 2b. We observe that the correlation is slightly more apparent and less noisy, which demonstrates the impact of uncertainty on LM generation process.

Weighted rule. The previous method assumes a primary uncertainty source for each sentence, which is not generic for real-world test cases that may contain multiple uncertainty sources. Instead, we measure the six defined uncertainty scores for a given text and assign a weight to each category by

[2]https://spacy.io/

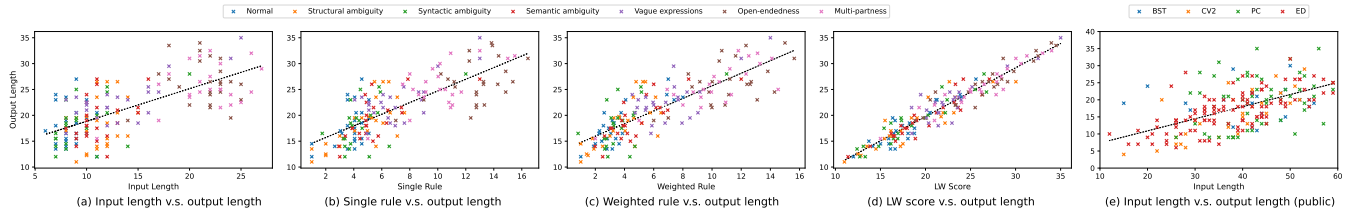


Fig. 2: Correlation between average output length across the five LMs and (a) input length, (b) single rule-based score, (c) weighted rule-based score, (d) LW model scores for self-generated sentences that contain different types of uncertainties, as well as (e) input length for sentences from the four benchmark datasets.

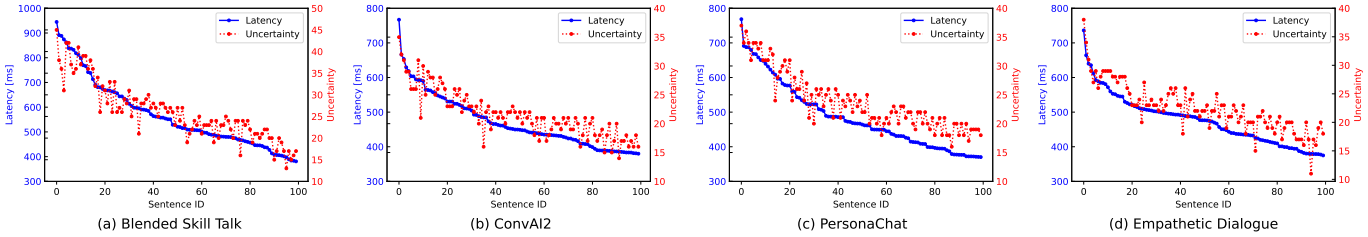


Fig. 3: Distribution of latency and corresponding uncertainty on four benchmark DG datasets, (a) *Blended Skill Talk*, (b) *ConvAI2*, (c) *PersonaChat*, (d) *Empathetic Dialogue*. The data points are ranked by descending order of latency.

learning a linear regression to the previously fitted line. We evaluate the correlation between weighted rule scores and output lengths for inputs with the corresponding type of uncertainty in Fig. 2c. We observe that the dependency between uncertainty scores and output lengths noticeably increases, without more data points getting close to the trend line.

Lightweight model. While hand-crafted rules can capture certain uncertainty for sentences, they are heuristic methods and not comprehensive enough since the data distribution is not learned. To make such estimation more reliable, we introduce a data-driven black-box lightweight (LW) multi-layer perceptron (MLP) [41] that takes the six rule-based scores as features and predicts the output length for any given query. Specifically, we train a LW model on the training sets of four benchmark datasets and evaluate the correlation between its predictions and output lengths for unseen queries in the test sets in Fig. 2d. We observe the output lengths are almost linearly dependent on our predicted scores, with only few noisy samples. We further evaluate the correlation between the predicted uncertainty scores and averaged inference latency across different LMs on the four benchmark datasets in Fig. 3. The predicted scores are highly consistent with the inference latencies across all datasets, i.e., sentences with smaller uncertainties generally require larger inference latencies. This suggests that our method can precisely estimate LM execution times for any unseen query in real-world dialogue scenarios.

C. System-level Optimization Opportunities

We now illustrate several precious system-level optimization ideas enabled by leveraging uncertainty score metric.

Prioritization. Online queries, though without intrinsic deadlines, have *priorities* (e.g., urgency of the task) that can be specified by `RT-LM` using the priority point parameter according to their estimated workloads. Leveraging the uncertainty score of each task (i.e., the estimated number of output tokens of each input), the scheduler shall make better prioritization decisions. Intuitively, prioritizing tasks that require shorter execution times and earlier priority points would improve throughput and timing correctness (often due to reduced head-of-line blocking), as illustrated in Fig. 4. In this example, five tasks that arrive at the same time (the length of each block presents its execution time) are scheduled by three strategies, namely Highest Priority Point First (HPF), Least Uncertainty First (LUF), and `RT-LM` utilizing Uncertainty-aware Prioritization (UP). As a result, HPF and LUF respectively miss two (J_4 and J_5) and three (J_2, J_4, and J_5) priority points, whereas UP misses only one priority point (J_2).

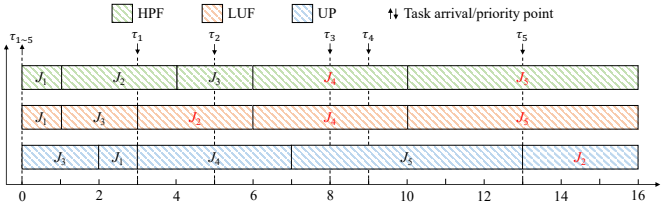


Fig. 4: Prioritization example for HPF, LUF, and UP. J_i denotes the i-th task, τ_i denotes its arrival/priority point. Tasks depicted in red color denote those missing their priority points.

Consolidation. In any heavily-loaded systems requiring machine learning workload multitasking, batch execution is a commonly-used method to enhance response time and timing correctness. Our estimated uncertainty scores can assist deciding which tasks shall be batched and executed together to better utilize hardware resources. Fig. 5 describes this idea using an intuitive example comparing two batch executions for eight tasks with a batch size of four. Fig. 5a presents a schedule under uncertainty-oblivious batching, e.g., HPF where tasks in

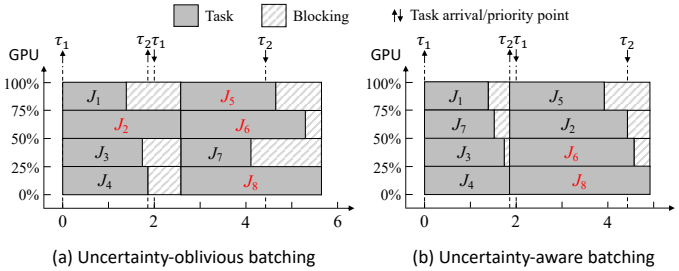


Fig. 5: Comparison of (a) random batching and (b) consolidation using uncertainty on eight tasks with a batch size of four. τ_i denotes the arrival/priority point for the i-th batch. Tasks with red notations miss the priority points.

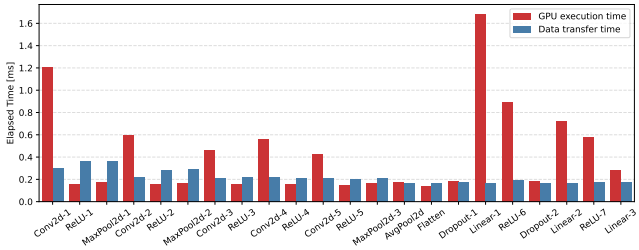


Fig. 6: Data transfer time (offloading) compared to GPU execution time for AlexNet.

each batch have similar priority points. Four tasks (J_2, J_5, J_6, J_8) miss priority points with a fairly low GPU utilization. Fig. 5b describes uncertainty-aware batching, where tasks in each batch have similar uncertainty scores. Only two tasks (J_6, J_8) miss priority points with an improved GPU utilization and shorter response time.

Strategic offloading to CPU. Previous works [42], [43] and our experiments indicate that offloading machine learning workloads to CPU cores often introduces non-negligible communication and synchronization overhead, negating the benefits of parallel utilization of both CPUs and GPUs. Fig. 6 depicts an illustrative example, where we compare the layer-wise data transfer cost with layer-wise GPU execution times for running AlexNet [44]. As seen, data transfer takes nearly the same amount of time as GPU execution for the majority of layers. Nonetheless, under overloaded situations or scenarios containing computation-demanding workloads, RT-LM could identify such tasks by checking whether the estimated uncertainty scores exceeds a pre-defined threshold. The scheduler can then decide whether offloading such demanding tasks to CPUs can improve the overall efficiency of the system. While it is likely that the negative impact due to offloading and communication can be totally negated by freeing up the precious GPU resource for executing other normal tasks, our intuition is to leverage uncertainty scores to reflect different levels of task demand and offload demanding tasks to CPU. This strategic offloading balances the workload between CPU and GPU, enabling efficient use of system resources and ensuring that the overall system remains responsive and productive.

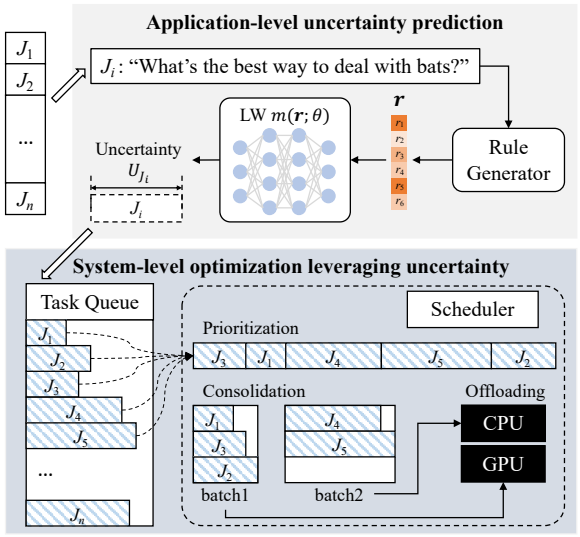


Fig. 7: Design overview of RT-LM.

IV. Design of RT-LM

A. Design Overview

In this section, we illustrate the overall design of RT-LM, as shown in Fig. 7. RT-LM comprises two major components: an application-level framework that quantifies task uncertainty, and a system-level framework that leverages this information for optimized scheduling (prioritization, dynamic consolidation) and resource allocation (strategic offloading).

Defined in Sec. III-B, the uncertainty score of an input text reflects the required output length and thus, its execution times. Leveraging this critical uncertainty information of input texts, RT-LM develops an uncertainty-aware system-level resource manager that makes better scheduling decisions. To ensure timing correctness, RT-LM introduces an uncertainty-aware priority scheduler that takes into account both uncertainty scores and priority points of tasks to reflect how critical a task is. By smartly considering both factors, RT-LM is capable of improving the system's throughput. Moreover, RT-LM includes a runtime consolidation mechanism to enhance the system's latency performance through uncertainty-aware batching. Our uncertainty estimation aids in deciding which tasks should be batched together to better utilize hardware resources. The system dynamically forms batches of tasks with similar execution times by reordering the tasks in two adjacent batches according to their uncertainty scores. In this way, tasks within each batch have both similar criticality and latency, leading to improved GPU utilization and less response time. Lastly, RT-LM integrates strategic CPU offloading to handle highly-demanding or malicious workloads. By leveraging uncertainty scores to indicate the demand of a task, RT-LM strategically offload tasks that may potentially lead to overloaded situations on GPUs to the CPU core to maintain a balanced workload distribution across the system.

B. Uncertainty-aware Prioritization

For any given input J, our rule generator RULEGEN$(\cdot)$ first yields a feature vector containing the intensity of the six linguistic uncertainties. Then our LW model m_θ takes the feature vector and predicts the final uncertainty score:

$$u_J = m_\theta(\text{RULEGEN}(J)) \tag{1}$$

In some scenarios such as conversational AI in healthcare [45], if an LM request has a user-specified deadline t_J, `RT-LM` can specify the priority point parameter using that deadline (d_J in Eq. 3 is replaced by t_J); whereas most LM-assisted dialogue systems do not have such user-specified deadlines. Based on our observations in Fig. 2e where longer inputs generally induce longer outputs, we empirically define a priority point for each task according to its input length $d_J = \varphi_f|J|$, where φ_f is a coefficient that projects input length to the latency of an LM f.

A straightforward way of factoring in both uncertainty and priority point into a system is to use the concept of "slack" (ζ), which measures the remaining time until the priority point:

$$p_J = \frac{1}{\zeta_J} = \frac{1}{d_J - r_J - \eta_f \cdot u_J} \tag{2}$$

Here r_J, d_J denote the arrival time and priority point of the task, respectively. The term u_J presents the uncertainty score of the task, reflecting the estimated output length, while η_f is a coefficient that projects output lengths to latencies, regarding the LM f. This slack-based approach prioritizes urgent tasks that are close to their priority points, which is suitable for systems with stringent priority point constraints and relatively stable task execution times.

However, for on-device LM systems facing workloads with high variability in uncertainties, such as input texts with a large uncertainty range causing LMs to generate outputs with varied lengths, a more flexible approach that can prioritize tasks with shorter execution times when needed ensures more predictable and consistent system performance. In `RT-LM`, we design Uncertainty-aware Prioritization (UP) where each task is assigned a priority p_J that reflects its weighted criticality:

$$p_J = \frac{1 - \alpha \cdot u_J}{d_J - r_J - \eta_f \cdot u_J} \tag{3}$$

Here α is a system-level hyper-parameter that provides a control over the impact of uncertainty on the priority. Specifically, $d_J - r_J - \eta_f \cdot u_J$ represents the estimated slack for the execution of the task, and $\alpha \cdot u_J$ is a scaled uncertainty score. The fraction computes the estimated execution time after considering the scaled uncertainty, normalized by how much time is left, to represent the criticality of a task. The intuition behind this priority assignment is that a task with a shorter slack window or smaller uncertainty score should have a higher priority. This ensures that tasks with imminent priority points or short execution times are attended to promptly, enhancing the likelihood of meeting their priority points. The factor α provides a level of adaptability to the system. A larger value of α implies that the system is placing a higher emphasis on tasks with lower uncertainties, regardless of how soon their priority points are, while a smaller α value reduces the impact of uncertainty on the priority calculation, placing a higher emphasis on the remaining time until the priority point. We search an optimal α value from 0 to 2.0 with a increment of 0.1 by testing the corresponding response time (see Fig. 13a).

C. Dynamic Consolidation

In the dynamic consolidation process, we aim to enhance the overall system efficiency by executing batches of tasks with similar estimated uncertainties, as they are more likely to have comparable processing requirements. The intuition is that executing tasks with similar workload characteristics as a batch can potentially lead to better resource utilization and reduced overheads, as illustrated in Fig. 5. Specifically, we maintain a queue of tasks sorted by the priority based on our UP algorithm (Eq. 3). We then group tasks with similar uncertainty scores together by introducing two hyper-parameters, λ and b. Among them, b determines the number of tasks to consider for a batch. Given a pre-defined batch size C, once the current batch accumulates $b \times C$ tasks from the task queue, we reorder these tasks according to their uncertainty scores. We then select the top-C tasks from this reordered list for execution. This mechanism ensures that tasks are executed in an order that prioritizes higher urgency as well as shorter execution times. Additionally, parameter λ controls the maximum allowable ratio in uncertainty scores between tasks within a batch. As we traverse the sorted list of tasks within the current batch, if we encounter a situation where the uncertainty score of the current task is more than λ times that of the previous one, we segment the list at this point. The tasks preceding this point are executed as a batch, while the remaining tasks are returned to the queue for future processing. The whole consolidation process unfolds as follows:

- Maintain a queue of tasks ordered by descending priority, based on the UP algorithm.
- Once accumulating $b \times C$ tasks in the current batch, reorder them in accordance with their uncertainty scores.
- Traverse the reordered batch of tasks. If the uncertainty of a task exceeds λ times the uncertainty of the previous task, or if the batch size C is met, segment the list at this point.
- Execute the tasks before the segmentation point as a batch, while returning the remaining tasks to the queue.

Dynamic consolidation provides flexibility in adjusting to varied workload characteristics and system conditions through the adjustment of the parameters b and λ. For instance, in scenarios where tasks exhibit diverse uncertainty scores, a smaller b or larger λ can be utilized to ensure that only tasks with similar uncertainties are grouped together. Conversely, if tasks have similar uncertainty scores, a larger b or smaller λ will form larger batches, potentially achieving higher system throughput. Moreover, dynamic consolidation can help balance the trade-off between throughput and predictability. By executing tasks with similar uncertainties as a batch, the system may exhibit more predictable behaviors, as estimating the execution

Algorithm 1 Uncertainty-aware framework of RT-LM

function UASCHED(α, λ, k, b);
Initialize a lightweight regressor m_θ;
for each $J \in \mathcal{D}_{train}$ **do** ▷ Offline profiling
  Rule scores $\mathbf{r}_J \leftarrow$ RULEGEN(J);
  LM output $y_J \leftarrow f(J)$ with length $|y_J|$;
  Minimize $\mathcal{L}_{MSE} \leftarrow \sum \{m_\theta(\mathbf{r}_J) - |y_J|\}^2$ and update the parameters of m_θ;
  Record the GPU util. under the current batch size;
Obtain the optimal batch size C_f for each LM $f(\cdot)$;
$\tau \leftarrow \text{quantile}_k(\{m_\theta(\text{RULEGEN}(J))|J \in \mathcal{D}_{train}\})$;
for each $(J, r_J, d_J) \in \mathcal{D}_{test}$ **do** ▷ Online scheduling
  Uncertainty score $u_J \leftarrow m_\theta(\text{RULEGEN}(J))$;
  Uncertainty-aware priority $p_J \leftarrow \frac{1-\alpha \cdot u_J}{d_J - r_J - \eta_f \cdot u_J}$;
  Put (p_J, u_J, J, r_J, d_J) in the task queue Q;
for each $(p_J, u_J, J, r_J, d_J) \in Q$ in descending p order **do**
  if $u_J > \tau$ **then**
    Offload J into the CPU batch; ▷ Offloading
  else
    Put J into a tmp batch $\mathcal{T}$;
  if $|\mathcal{T}| = \lfloor b \cdot C_f \rfloor$ **then** ▷ Consolidation
    Sort $J \in \mathcal{T}$ in ascending uncertainty u order;
    $u_{prev} \leftarrow \mathcal{T}[0]$; count $\leftarrow 0$;
    while $u_J \leq \lambda \cdot u_{prev} \vee$ count $< C_f$ **do**
      $u_{prev} \leftarrow u_J$; count++;
    Append $\mathcal{T}$[: count] into the GPU batch;
    Put $\mathcal{T}$[count :] back into the task queue Q;
    Clear the tmp batch $\mathcal{T}$.

time of a batch is often simpler than predicting individual task execution times. Meanwhile, by executing tasks in batches, the system can potentially achieve higher throughput compared to executing tasks individually.

D. Strategic Offloading to CPU

In the dynamic consolidation process described above, tasks are assigned to batches and then executed based on uncertainty scores. However, such a process can lead to the situation where some tasks with high uncertainty scores (e.g., malicious, adversarial tasks) may potentially delay the execution of the whole batch, negatively affecting the overall system performance. To address this, we propose a protective mechanism, termed 'strategic offloading', to offload potentially malicious tasks and execute them separately on CPU cores.

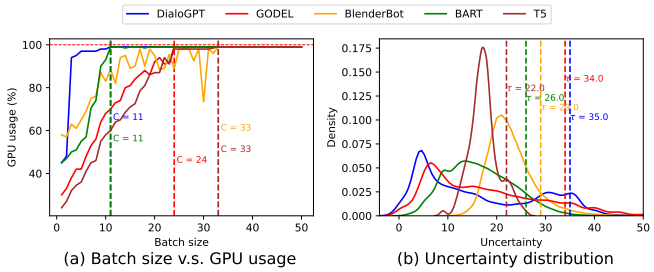
In our implementation, we define a parameter k ($0 < k < 1$) which denotes the top-k percentage of uncertainty scores in the training set to control the malicious threshold τ:

$$\tau = \text{quantile}_k(\{m_\theta(\text{RULEGEN}(J))|J \in \mathcal{D}_{train}\}) \quad (4)$$

In essence, τ corresponds to the boundary of the highest k-percentile of uncertainty scores. If the uncertainty score of a task is larger than τ, it is offloaded to a CPU batch for separate execution. Otherwise, it is assigned to a GPU batch

TABLE II: Hardware platforms used in our experiments.

	Edge Server	NVIDIA AGX Xavier
CPU	96-core AMD EPYC 7352 24-Core Processor	8-core NVIDIA Carmel Armv8.2 64-bit CPU
GPU	NVIDIA RTX A4500	NVIDIA Volta GPU
Memory	512GB	16GB LPDDR4x
Storage	8TB SSD	32GB eMMC



(a) Batch size v.s. GPU usage (b) Uncertainty distribution

Fig. 8: Offline decisions on (a) optimal batch size C and (b) malicious threshold τ ($k = 0.9$) for the five LMs.

for grouped execution. Furthermore, we ensure that there is always a batch of tasks ready for execution. If the task queue is empty and there are remaining tasks in the GPU batch, these tasks are offloaded for execution. Similarly, if there are no tasks in the GPU and CPU batches, the remaining tasks from the task queue are offloaded to the appropriate execution batch based on their uncertainty scores. This strategic offloading mechanism provides a layer of protection against extreme execution times, ensuring malicious tasks do not excessively delay the execution of a batch and promising a more predictable and reliable system performance, particularly under workloads with high variability. By carefully controlling the offloading parameter k, this mechanism can be tuned to balance the benefits of grouping tasks for efficient execution against the potential delays caused by malicious tasks.

E. Pseudo Code and Illustration

Algorithm 1 illustrates the whole framework of RT-LM, known as UASCHED. It takes several aforementioned control parameters, α, λ, k, and b, and operates in two main phases: offline profiling and online scheduling.

Offline profiling. The algorithm starts by initializing a LW regressor m_θ. For each task in the training set, RULEGEN($\cdot$) generates rule scores $\mathbf{r}_J$, which is taken by m_θ as features and calculates the output length from the LM. The algorithm then minimizes the Mean Squared Error (MSE) between the estimated output lengths and the LM output lengths, thereby updating the LW model. It also records GPU utilization to determine the minimum batch size C_f for the LM $f(\cdot)$ that can better utilize hardware resources, e.g., when GPU usage reaches 100%. Finally, it determines the malicious threshold τ according to the uncertainty score distribution.

Online scheduling. The algorithm iterates over tasks in the test set, calculating uncertainty scores using the pre-trained LW model m_θ, and then placing them into a task queue. The

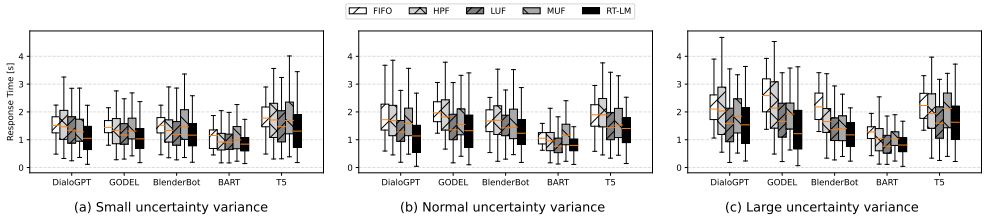


Fig. 9: The distribution of response time across five LMs for sentences with (a) small, (b) normal, and (c) large uncertainty variance on the edge server.

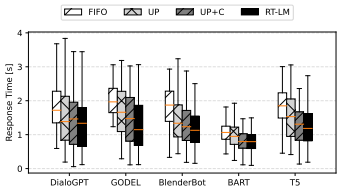


Fig. 10: Ablation study of response time on the edge server.

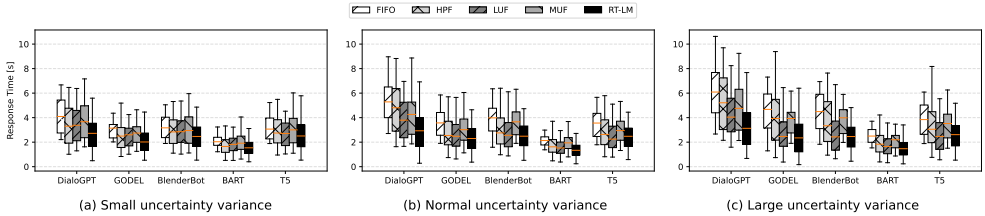


Fig. 11: The distribution of response time across five LMs for sentences with (a) small, (b) normal, and (c) large uncertainty variance on the AGX Xavier.

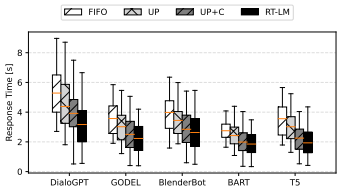


Fig. 12: Ablation study of response time on the AGX Xavier.

tasks are then popped and processed in a descending order of priority scores. If a task's uncertainty score is greater than the threshold, it is offloaded to a CPU batch; otherwise, it is placed in a temporary batch. If the temporary batch reaches a size of $b \cdot C_f$, the scheduler sorts tasks in the batch in ascending order of uncertainty scores. It then segments the batch at a point where the current uncertainty score is larger than λ times that of the previous one or if the pre-defined batch size C_f has been reached. The segmented tasks are offloaded to a GPU batch, while the remaining ones are put back into the queue.

V. Implementation and Evaluation

A. Experiment Setup

Testbeds. We implement `RT-LM` and conduct an extensive set of experiments on an edge server, as shown in Table II, simulating the single-device multitasking scenarios of online chatbots or services, and live-translation services.

Benchmark. We evaluate `RT-LM` across five state-of-the-art LMs that are widely used in dialogue systems—DialoGPT [25], GODEL [26], BlenderBot [27], BART [28], and T5 [29]—on four benchmark datasets: *Blended Skill Talk* [30], *PersonaChat* [31], *ConvAI2* [32], and *Empathetic Dialogues* [33]. We use the pre-trained versions of these models—*DialoGPT-medium*, *GODEL-v1_1-base-seq2seq*, *blenderbot-400M-distill*, *bart-base*, *t5-base* and annotated datasets released by Hugging Face[3].

Metrics. We evaluate `RT-LM`'s performance w.r.t. the average response time, throughput, and runtime overhead. We also delve deeper into the effect of different components of `RT-LM` on the system-level performance, the robustness of `RT-LM` against different parameter settings, and its effectiveness under different proportions of malicious tasks.

Hyper-parameters. For the offline profiling, we initialize a lightweight MLP which has four layers of hidden size [100, 200, 200, 100], and train the model with a learning rate of 1e-4. We record the average GPU usage for the five LMs with different batch sizes in Fig. 8a. Specifically, we choose an optimal batch size (i.e., minimum batch size that a LM can reach 100% GPU usage) of 11, 24, 33, 11, 33, for DialoGPT, GODEL, BlenderBot, BART, T5, respectively. We further record the distribution of uncertainty scores for each LM in Fig. 8b, and select a malicious threshold of 35, 34, 29, 26, 22 for DialoGPT, GODEL, BlenderBot, BART, T5, respectively. We set the uncertainty-weight α as 1.0, the output-latency coefficients η as 0.05, 0.04, 0.1, 0.05, 0.04, and the input-latency coefficients φ as 0.08, 0.10, 0.13, 0.08, 0.07 across the five LMs for priority assignment; λ, b as 1.5, 1.8, respectively for dynamic consolidation; and k as 0.9 for protective mechanism. To gather necessary statistics, we employ the tegrastats utility for recording GPU and CPU memory usage. Additionally, we use Python's time library to track the arrival and end time of each task, as well as the latency incurred by `RT-LM`.

Workload setup. Real-world human-generated processes, such as phone calls to a call center, can often be represented as a Poisson process, where the number of arrivals within a specific time interval is governed by a Poisson distribution [46], [47]. Given the independent nature of user queries in our context, we adopt a similar model to simulate task arrivals. This model is principally defined by its average arrival rate, denoted as β (representing queries per minute). We generated synthetic traces by sampling inter-arrival times from an exponential distribution with differing mean $\mu = \frac{1}{\beta}$ to modulate the arrival rate. To create time-varying synthetic workloads, we continuously evolve the workload generator across different exponential distributions throughout the process. This involves iterating through integer values of β ranging from 10 to 150. For each minute, we sample from the corresponding exponential distribution, ensuring a comprehensive representation

[3]https://huggingface.co/

TABLE III: Maximum response time (s) and percentage of improvement for sentences with small, normal, and large uncertainty variance on the edge server. The evaluated methods consist of uncertainty-oblivious (former) and uncertainty-aware (latter) ones. **Bold** numbers denote the best metric values among them.

Method	DialoGPT			GODEL			BlenderBot			BART			T5		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
FIFO	2.25	3.75	3.90	**2.15**	3.06	3.93	**2.24**	2.52	3.41	**1.87**	1.93	1.95	**2.90**	2.95	3.30
HPF	3.25	3.92	4.68	2.75	3.79	4.53	2.90	3.54	3.37	2.34	2.13	2.63	3.56	4.13	3.97
LUF	2.85	**2.77**	3.55	2.47	3.06	3.41	2.86	2.79	2.93	1.98	1.82	2.19	3.24	3.43	**3.17**
MUF	3.03	3.68	3.93	3.52	3.74	4.21	3.36	3.52	3.10	2.97	3.00	2.38	4.01	3.30	3.98
RT-LM	**2.24**	2.96	**3.18**	2.52	**2.80**	**3.17**	2.92	**2.26**	**2.38**	1.93	**1.66**	**1.86**	3.45	**2.64**	3.25
	-0.4%	-21.1%	-18.5%	+17.2%	-8.5%	-19.3%	+30.4%	-10.3%	-30.2%	+3.2%	-14.0%	-4.6%	+19.0%	-23.4%	-1.5%

of workload scenarios, from light-load phases to high-traffic peaks. Following the generation of these traces, we shuffle the test dataset and map them to the created arrival patterns. To enhance realism, acknowledging that users may require some time to complete a query, we introduced a wait time interval $\xi = 2$ seconds so that tasks arriving within this span are processed as either a single batch or multiple batches[4].

B. Latency Performance

We evaluate the latency performance of various strategies by calculating their response time – the time elapsed between a task's end time and its arrival time across the five LMs. Naturally, a lower average response time indicates a more efficient system. We compare RT-LM to the following baselines:

- First-In-First-Out (FIFO): Tasks are queued based on their arrival times, creating uncertainty-oblivious random batches with a fixed size for execution.
- Highest Priority-Point First (HPF) [48]: Tasks with higher priority points are prioritized. This approach batches tasks with similar priority points together, maintaining a fixed batch size, yet remains uncertainty-oblivious.
- LUF: Tasks with lower uncertainty scores are given precedence. Those with comparable uncertainty scores (or execution times) are batched together using a fixed size.
- Maximum Uncertainty First (MUF): This strategy prioritizes tasks with higher uncertainty scores. Those with analogous scores are batched together with a set size.

To gauge the impact of uncertainty on system-level performance, we evaluate all methods across three subsets of tasks featuring small, medium, and large variance of uncertainty scores on the edge server. Fig. 9 demonstrates the distribution of response time values, while Table III records the worst-case response time for each method across task subsets. From our observations: 1) Uncertainty-aware strategies tend to surpass uncertainty-oblivious ones, especially when input data exhibits varied uncertainty scores. For the small-variance subset, all methods display similar response times in Fig. 9a, with the maximum values of LUF, MUF, RT-LM even larger than FIFO, HPF in some cases in Table III, but on the large-variance subset, LUF, MUF, RT-LM consistently outperform FIFO and HPF. This is because when tasks exhibit similar workloads, all strategies essentially mimic FIFO. However, when there's significant variance in task uncertainty, grouping tasks with analogous uncertainty scores reduces the likelihood of computation-intensive tasks holding up the entire batch. 2) Generally, LUF produces a better performance than MUF. By prioritizing tasks with high uncertainty, MUF can inadvertently cause the entire system to lag, thus compromising average response times. 3) RT-LM consistently exhibits superior performance, achieving the most efficient response times across all LMs. The average response time of RT-LM is roughly 0.8s less than FIFO for BART in Fig. 9c; and its maximum response time is up to 30% smaller than FIFO for BlenderBot in Table III. This suggests that considering both execution times and priority points in task prioritization can further optimize latency performance. This dual consideration ensures RT-LM is versatile across varied workload distributions. 4) Larger LMs are more sensitive to variations in task uncertainty, requiring even more execution times for tasks with high uncertainty scores, thereby benefiting more from uncertainty-aware strategies, e.g., RT-LM improves the maximum response time over FIFO to a larger extent for GODEL and BlenderBot (20% and 30%) than other LMs.

C. Throughput Performance

We further evaluate the throughput of various strategies as the average completed tasks per minute, across the five LMs, on the edge server. As expected, a higher throughput implies a more efficient system. Table IV summarizes the results on the three subsets. We observe the throughput profiles of all methods are highly consistent with their latency performance metrics. Specifically, uncertainty-aware strategies notably exhibit larger advantages over uncertainty-oblivious ones when the uncertainty variance of test inputs grows, e.g., RT-LM can process over 6 more tasks per minute than FIFO, with DialoGPT in the large-variance subset. Among these, LUF is generally superior to MUF. RT-LM, however, stands out by consistently outperforming all other strategies. Moreover, uncertainty-aware strategies, particularly on larger LMs, can significantly boost system efficiency, e.g., RT-LM boosts the average throughput by 10% to 30% for BART and GODEL.

D. Ablation Study

To elucidate the superiority of RT-LM, we conduct an ablation study investigating the individual contributions of each component of our method to the response time and throughput performance:

[4]We conducted supplementary experiments using diverse sets of μ and ξ values. The findings consistently align with the trends observed in Fig.9$\sim$11.

TABLE IV: Average throughput for sentences with small, normal, and large uncertainty variance on the edge server.

Method	DialoGPT			GODEL			BlenderBot			BART			T5		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
FIFO	21.68	18.00	15.68	17.89	18.28	13.36	21.15	17.90	17.63	32.30	30.62	25.92	17.86	16.57	16.15
HPF	20.26	19.27	16.41	19.55	18.69	13.99	21.26	18.28	17.32	**33.59**	30.22	26.56	18.10	16.75	17.18
LUF	23.19	21.09	19.97	19.71	19.17	17.68	**21.34**	19.48	19.81	32.86	31.02	28.52	19.75	18.94	18.84
MUF	22.40	20.06	19.44	19.14	18.34	16.76	21.20	18.92	20.08	32.03	31.28	27.97	19.58	17.78	17.52
RT-LM	**24.61**	**23.89**	**22.34**	**23.73**	**21.54**	**19.78**	21.12	**20.66**	**20.80**	32.14	**31.71**	**28.64**	**22.28**	**21.94**	**20.03**

- Uncertainty-aware prioritization (UP): We compare uncertainty-oblivious prioritization strategies, namely FIFO and HPF, with UP for response time and throughput evaluation, respectively.
- Dynamic consolidation: We contrast UP (using static batching) with its dynamic consolidation counterpart (UP+C).
- Strategic offloading: We compared UP+C with RT-LM, which facilitates execution of malicious tasks on the CPU.

Fig. 10 illustrates the subtle improvements of each component-enabled method over its component-oblivious counterpart in terms of reduced response times on the edge server, with RT-LM consistently outperforming the rest. For example, UP achieves an average response time of 0.2∼0.7s less than FIFO. This indicates all three components of RT-LM are integral to its superior performance. Notably, the performance boost derived from prioritization and consolidation is typically larger than offloading, e.g., the average response time gap between UP+C and RT-LM is smaller than other pairs in most cases. This suggests that our prioritization and consolidation are more consequential in improving efficiency. Interestingly, strategic offloading has slightly more significant impact on larger LMs, e.g., RT-LM reduces the average response time over UP+C to 0.4s for GODEL, while their performance are nearly the same for BART. This is because computational demanding tasks have larger impact on sophisticated LMs, causing even more severely overloaded systems.

E. On-Device Evaluation

Emerging embedded devices, augmented with powerful computing capabilities and LM intelligence [49], have the potential to serve as the local central service in future smart homes. These devices may support hundreds of IoT devices, facilitating concurrent multi-user or multi-device (e.g., refrigerator, air conditioner) communications with a single LM, a concept known as connected intelligence. In this context, we delve into the performance evaluation of various methods on an NVIDIA AGX Xavier (see Table II), which is widely used in various applications such as autonomous driving [50], [51] and robotics [52]–[55], to reflect the feasibility of RT-LM in on-device multitasking scenarios.

Fig. 11 showcases the response time of all evaluated methods across three subsets on the AGX Xavier. The observed patterns largely mirror those seen on the edge server. For instance, uncertainty-aware strategies excel, particularly in subsets with diverse uncertainty characteristics. LUF is generally more efficient than MUF, RT-LM consistently outperforms other baselines across all LMs, and uncertainty-aware strategies

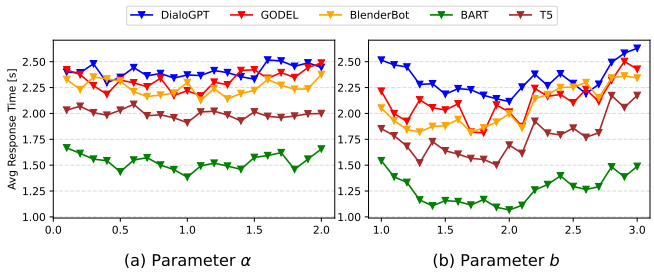


Fig. 13: Study of average response time with different values of (a) α and (b) b across five LMs on the edge server.

derive greater efficiency benefits from larger LMs, such as GODEL. Furthermore, a comparative analysis between the two platforms reveals an interesting insight: high-performance devices, being quicker in execution, tend to display a smaller disparity in performance across different methods compared to embedded devices. This subtly hints at a diminished relative advantage for RT-LM on more powerful devices.

Fig. 12 depicts the individual contributions of each RT-LM component, in terms of reduced response time on the embedded device. The findings align with on the edge server: all three components collectively boost its performance, prioritization and consolidation emerge as more influential factors in enhancing efficiency than offloading, and larger LMs generally derive more pronounced benefits from offloading.

F. Parameter Study

We explore the impact of two key hyperparameters, α and b, which control the influence of uncertainty in priority computation and the batch size determined by the number of tasks, on RT-LM. We vary α from 0.1 to 2.0 (with a fixed $b = 2.0$) and b from 1.0 to 3.0 (with a fixed $\alpha = 1.0$), incrementing by 0.1 in both cases, and assess the resulting average response time of RT-LM across different LMs.

Fig. 13a shows that RT-LM is robust to changes in α, with a maximum divergence in response time not exceeding 0.35s for each LM. This resilience indicates that UP functions as a well-balanced, uncertainty-aware priority, aptly mediating between priority points and execution times for tasks. An optimal α value of 1.0 is indicated by our performance metrics. Placing a higher emphasis on either uncertainty (larger α) or remaining time until the priority point (smaller α) results in a slight increase of response time.

Fig. 13b reveals that b has a more significant impact on latency performance than α, with the maximum deviation in

TABLE V: An example of crafted sentence that causes DialoGPT to generate much longer outputs. *Italics* and ~~strike through~~ denote added and removed tokens, respectively.

Q: Not really. Let's ~~talk~~ *think* about food. What do you like to eat? I ~~love~~ *like* fish.
A: I love fish too! What is your favorite kind?
Â: I like to eat fish too. What is your favorite kind? I like pasta, filipino, steak, etc. I talk a lot on IRC and it is fun to learn about it with some other guys.

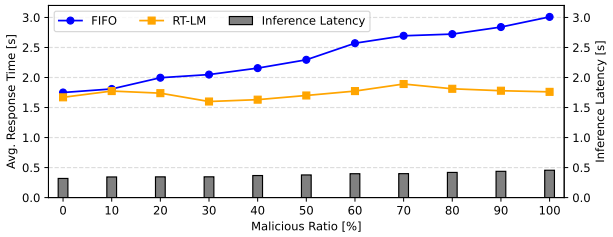


Fig. 14: Average response time and LM inference latency on the edge server, under varying ratios of malicious tasks.

response time reaching about 0.75s for T5. This indicates a considerable dependence of dynamic consolidation on the number of tasks considered for a batch. Optimal performance is achieved at $b = 1.8$. Values below or above this introduce inefficiencies, either mimicking static batching or causing delays in task completion due to longer wait time.

G. Evaluating Malicious Scenarios

To evaluate the robustness of RT-LM against malicious inputs, we apply a state-of-the-art adversarial attack method [56] that crafts provided input texts to elongate LM outputs. Table V presents an example of a malicious sentence designed to prompt an LM to generate longer output $\hat{\mathbf{A}}$ than the original one $\mathbf{A}$, leading to a computational burst and degraded system performance. Tasks are deemed malicious if their uncertainty scores exceed a predefined threshold (see Eq. 4). To assess the response, we control the proportion of deliberately crafted malicious tasks within a range of 0% to 100%, increasing in increments of 10%, and evaluate subsequent system latency performance.

Fig. 14 shows the effects of varying ratios of malicious tasks on the average response time of both FIFO and RT-LM, as well as the associated average inference latency across different LMs. As seen, RT-LM is proficient in managing extreme conditions wherein a large proportion of malicious tasks need to be processed, outperforming the uncertainty-oblivious FIFO. When the malicious task ratio exceeds 30%, FIFO exhibits high sensitivity, with the average response time increases from around 2.0s to 3.0s. Whereas RT-LM is resilient against malicious tasks, maintaining a steady average response time of around 1.5∼1.9s. Our results confirm that RT-LM effectively prevents malicious tasks from hindering the execution of other critical tasks. This resilience enhances RT-LM's suitability for applications like chatbots [57], personal assistants [58], and conversational AI in healthcare [45] where defense against adversarial attacks is crucial.

TABLE VI: Latency and memory of offline profiling.

LM	Total LW latency (s)		Memory
	Train	Ratio	Train
DialoGPT	351	3.01%	14,607 MB
GODEL	490	3.96%	14,768 MB
BlenderBot	448	3.71%	14,723 MB
BART	392	3.25%	14,631 MB
T5	369	3.06%	14,639 MB

TABLE VII: Latency, memory, and CPU/GPU utilization of online scheduling. Prior., consol., and off. denote prioritization, consolidation, and offloading.

LM	Avg. per-task latency (ms)				Memory	CPU / GPU util.
	Prior.	Consol.	Off.	Ratio	Test	Ratio
DialoGPT	8.04	0.42	0.37	2.10%	11,293 MB	97% / 92%
GODEL	7.78	0.43	0.49	2.04%	12,795 MB	93% / 97%
BlenderBot	9.24	0.53	0.40	2.39%	12,136 MB	99% / 95%
BART	7.84	0.35	0.10	2.06%	11,979 MB	97% / 91%
T5	8.39	0.33	0.18	2.27%	11,653 MB	95% / 90%

H. Overhead Analysis

Analyzing overhead is crucial in practical real-time systems which are more complicated and variant. A solution with high overhead may undermine response time and throughput, as the scheduling process may severely block task execution. We present an analysis of both latency and memory usage introduced by RT-LM on the edge server, offering insights into the practical efficiency of our design.

Offline Profiling. We initialize an LW model and train it for 100 epochs, using the LM outputs as ground truths. We report both the average training time per epoch and its proportion relative to the LM inference time. Memory usage during this phase is also recorded. As shown in Table VI, our training consumes merely around 3∼4% of the LM inference latency, and less than 3% of the total available memory (512 GB), demonstrating the overhead efficiency of RT-LM.

Online Scheduling. We evaluate the average per-task latency of each component of RT-LM and compare the combined latency to the LM inference time. We also record the average memory usage as well as CPU/GPU utilization during online scheduling. Table VII reveals that RT-LM introduces less than 3% additional latency overhead relative to the LM inference time (around 415 milliseconds per task). Such small overheads are unlikely to affect real-time dialogue systems noticeably. Notably, prioritization accounts for the majority of scheduling time, as uncertainty is computed and queued at this stage. For all LMs, CPU/GPU utilization reach over 90%, which suggests effective resource allocation under RT-LM.

VI. Related Work and Discussion

Real-time DNN Inference. Recent research has improved real-time Deep Neural Network (DNN) performance with strategies optimizing performance-accuracy trade-offs [59]–[61], and exploring system design for DNN execution [42], [43], [62]–[67]. Despite these advancements, previous works

neither consider the dynamics of DNNs for different execution times of inputs. In contrast, our proposed method, RT-LM, builds upon these existing scheduling algorithms by incorporating uncertainty estimation to further enhance performance and resource allocation.

Uncertainty Estimation. Uncertainty estimation has been a topic of interest in the machine learning and NLP community, particularly in the context of deep learning [68]. Methods like Monte Carlo dropout [23] and Bayesian neural networks [69] have been proposed to quantify the uncertainty in model predictions. Previous works [17], [39], [70] also show that uncertainty may cause an LM to generate outputs with varied lengths. Our method employs a lightweight regressor to estimate the uncertainty in terms of the output length of an LM inference, which can be used to inform the scheduling process, improving resource utilization and response time.

Intelligent Edge Server Systems. In cloud-edge-client hierarchical systems, AI models are co-deployed on the cloud and edge servers [49], [71], where multiple requests from diverse users via edge devices can be processed concurrently by the DNNs. Notable examples of such applications include online chatbots and live translation services. Additionally, cloud servers frequently grapple with load balancing across multiple workers [72]. RT-LM could prioritize critical requests and redirect malicious tasks to CPU cores, thereby enhancing overall system performance and reducing the threat of performance attacks against DNNs [56], [73]–[77].

Limitations of RT-LM. RT-LM mainly targets system-level optimization in heavy-workload scenarios, emphasizing concurrent task processing by taking into account the uncertainty characteristics of each task. In real-world on-device LM-embedded systems, where queries typically arrive sequentially, there's room for further improvement, e.g., optimizing performance for each individual task by leveraging the correlation between uncertainty and layer-level LM inference/training efficiency could be pursued. Additionally, our current approach is designed for single-machine scenarios. Expanding to hybrid deployment setups, such as server-edge combinations, is an avenue worth exploring. Moreover, RT-LM doesn't account for memory and power constraints, which could cause potential out-of-memory (OOM) issues on edge environments and pose challenges when deploying on low-power devices. Although deep learning compilers [78], [79] may mitigate the challenges posed by limited resources in such scenarios, adapting RT-LM to work efficiently in memory-constrained edge settings and optimizing LM inference from a power-efficiency standpoint is an area yet to be addressed.

VII. Conclusion

In this paper, we introduced RT-LM, a novel uncertainty-aware resource management for real-time on-device LMs. Our extensive evaluations demonstrated the superior performance of RT-LM in terms of response time, system throughput, and robustness to various system settings, while maintaining low overhead and excellent memory efficiency. In the future, we will focus on further optimizing the uncertainty estimation mechanism and expanding the applicability of RT-LM to more diverse and dynamic real-world workloads.

Acknowledgment

This research was supported by the National Science Foundation under Grants CNS Career 2230968, CPS 2230969, CNS 2300525, CNS 2343653, CNS 2312397.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[2] Y. Chen, Y. Zhang, C. Zhang, G. Lee, R. Cheng, and H. Li, "Revisiting self-training for few-shot learning of language model," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 9125–9135. [Online]. Available: https://aclanthology.org/2021.emnlp-main.718

[3] Y. Chen, Y. Zhang, B. Wang, Z. Liu, and H. Li, "Generate, discriminate and contrast: A semi-supervised sentence representation learning framework," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 8150–8161. [Online]. Available: https://aclanthology.org/2022.emnlp-main.558

[4] S. Li, Y. Li, J. Ni, and J. McAuley, "SHARE: a system for hierarchical assistive recipe editing," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 11 077–11 090. [Online]. Available: https://aclanthology.org/2022.emnlp-main.761

[5] R. Janssens, P. Wolfert, T. Demeester, and T. Belpaeme, "Cool glasses, where did you get them?: Generating visually grounded conversation starters for human-robot dialogue," in *ACM/IEEE International Conference on Human-Robot Interaction, HRI 2022, Sapporo, Hokkaido, Japan, March 7 - 10, 2022*, D. Sakamoto, A. Weiss, L. M. Hiatt, and M. Shiomi, Eds. IEEE / ACM, 2022, pp. 821–825.

[6] J. Yang, P. Wang, Y. Zhu, M. Feng, M. Chen, and X. He, "Gated multimodal fusion with contrastive learning for turn-taking prediction in human-robot dialogue," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*. IEEE, 2022, pp. 7747–7751.

[7] M. Kraus, N. Wagner, W. Minker, A. Agrawal, A. Schmidt, P. K. Prasad, and W. Ertel, "KURT: A household assistance robot capable of proactive dialogue," in *ACM/IEEE International Conference on Human-Robot Interaction, HRI 2022, Sapporo, Hokkaido, Japan, March 7 - 10, 2022*, D. Sakamoto, A. Weiss, L. M. Hiatt, and M. Shiomi, Eds. IEEE / ACM, 2022, pp. 855–859.

[8] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, 2022.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008.

[10] C. Xu and J. McAuley, "A survey on model compression for natural language processing," *arXiv preprint arXiv:2202.07105*, 2022.

[11] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[12] D. Gao, X. He, Z. Zhou, Y. Tong, K. Xu, and L. Thiele, "Rethinking pruning for accelerating deep inference at the edge," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 155–164.

[13] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[14] B. J. Grosz, A. K. Joshi, and S. Weinstein, "Centering: A framework for modeling the local coherence of discourse," *Computational Linguistics*, vol. 21, no. 2, pp. 203–225, 1995.

[15] D. Loureiro and A. Jorge, "Language modelling makes sense: Propagating representations through WordNet for full-coverage word sense disambiguation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 5682–5691.

[16] C. Gunasekara, G. Feigenblat, B. Sznajder, R. Aharonov, and S. Joshi, "Using question answering rewards to improve abstractive summarization," in *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 518–526.

[17] A. Fader, L. Zettlemoyer, and O. Etzioni, "Open question answering over curated and extracted knowledge bases," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1156–1165.

[18] M. Zhu, A. Ahuja, D.-C. Juan, W. Wei, and C. K. Reddy, "Question answering with long multiple-span answers," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 3840–3849.

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 3104–3112.

[20] Z. Li, J. Cai, S. He, and H. Zhao, "Seq2seq dependency parsing," in *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 3203–3214.

[21] J. He, X. Zhang, S. Lei, Z. Chen, F. Chen, A. Alhamadani, B. Xiao, and C. Lu, "Towards more accurate uncertainty estimation in text classification," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 8362–8372.

[22] L. Kong, H. Jiang, Y. Zhuang, J. Lyu, T. Zhao, and C. Zhang, "Calibrated language model fine-tuning for in- and out-of-distribution data," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1326–1340.

[23] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, M. Balcan and K. Q. Weinberger, Eds., vol. 48. JMLR.org, 2016, pp. 1050–1059.

[24] W. Niu, Z. Kong, G. Yuan, W. Jiang, J. Guan, C. Ding, P. Zhao, S. Liu, B. Ren, and Y. Wang, "Real-time execution of large-scale language models on mobile," *arXiv preprint arXiv:2009.06823*, 2020.

[25] Y. Zhang, S. Sun, M. Galley, Y.-C. Chen, C. Brockett, X. Gao, J. Gao, J. Liu, and B. Dolan, "DIALOGPT : Large-scale generative pre-training for conversational response generation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, Jul. 2020, pp. 270–278.

[26] B. Peng, M. Galley, P. He, C. Brockett, L. Liden, E. Nouri, Z. Yu, B. Dolan, and J. Gao, "GODEL: large-scale pre-training for goal-directed dialog," *CoRR*, vol. abs/2206.11309, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2206.11309

[27] S. Roller, E. Dinan, N. Goyal, D. Ju, M. Williamson, Y. Liu, J. Xu, M. Ott, E. M. Smith, Y.-L. Boureau, and J. Weston, "Recipes for building an open-domain chatbot," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, Apr. 2021, pp. 300–325.

[28] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 7871–7880.

[29] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[30] E. M. Smith, M. Williamson, K. Shuster, J. Weston, and Y.-L. Boureau, "Can you put it all together: Evaluating conversational agents' ability to blend skills," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 2021–2030.

[31] S. Zhang, E. Dinan, J. Urbanek, A. Szlam, D. Kiela, and J. Weston, "Personalizing dialogue agents: I have a dog, do you have pets too?" in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 2204–2213.

[32] E. Dinan, V. Logacheva, V. Malykh, A. Miller, K. Shuster, J. Urbanek, D. Kiela, A. Szlam, I. Serban, R. Lowe *et al.*, "The second conversational intelligence challenge (convai2)," in *The NeurIPS'18 Competition*. Springer, 2020, pp. 187–208.

[33] H. Rashkin, E. M. Smith, M. Li, and Y. Boureau, "Towards empathetic open-domain conversation models: A new benchmark and dataset," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, A. Korhonen, D. R. Traum, and L. Màrquez, Eds. Association for Computational Linguistics, 2019, pp. 5370–5381.

[34] A. Shelmanov, E. Tsymbalov, D. Puzyrev, K. Fedyanin, A. Panchenko, and M. Panov, "How certain is your Transformer?" in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, Apr. 2021, pp. 1833–1840.

[35] M. Wu, N. S. Moosavi, D. Roth, and I. Gurevych, "Coreference reasoning in machine reading comprehension," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 5768–5781.

[36] K. Lee, L. He, M. Lewis, and L. Zettlemoyer, "End-to-end neural coreference resolution," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 188–197.

[37] J. Hirschberg and D. J. Litman, "Empirical studies on the disambiguation of cue phrases," *Comput. Linguistics*, vol. 19, no. 3, pp. 501–530, 1993.

[38] L. Lebanoff and F. Liu, "Automatic detection of vague words and sentences in privacy policies," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3508–3517.

[39] A. Mao, N. Raman, M. Shu, E. Li, F. Yang, and J. Boyd-Graber, "Eliciting bias in question answering models through ambiguity," in *Proceedings of the 3rd Workshop on Machine Reading for Question Answering*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 92–99.

[40] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 1321–1330.

[41] H. Ramchoun, Y. Ghanou, M. Ettaouil, and M. A. Janati Idrissi, "Multilayer perceptron: Architecture optimization and training," 2016.

[42] W. Kang, K. Lee, J. Lee, I. Shin, and H. S. Chwa, "Lalarand: Flexible layer-by-layer CPU/GPU scheduling for real-time DNN tasks," in *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*. IEEE, 2021, pp. 329–341.

[43] M. Ji, S. Yi, C. Koo, S. Ahn, D. Seo, N. D. Dutt, and J. Kim, "Demand layering for real-time DNN inference with minimized memory usage," in *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*. IEEE, 2022, pp. 291–304.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held*

December 3-6, 2012, Lake Tahoe, Nevada, United States, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114.

[45] U. Bharti, D. Bajaj, H. Batra, S. Lalit, S. Lalit, and A. Gangwani, "Medbot: Conversational artificial intelligence powered chatbot for delivering tele-health after covid-19," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2020, pp. 870–875.

[46] E. Cinlar, *Introduction to stochastic processes*. Courier Corporation, 2013.

[47] S. M. Ross, *Introduction to probability models*. Academic press, 2014.

[48] J. W. Liu, "Real-time systems," 2000.

[49] Y. Shen, J. Shao, X. Zhang, Z. Lin, H. Pan, D. Li, J. Zhang, and K. B. Letaief, "Large language models empowered autonomous edge AI for connected intelligence," *CoRR*, vol. abs/2307.02779, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.02779

[50] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[51] B. Kisačanin, "Deep learning for autonomous vehicles," in *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2017, pp. 142–142.

[52] A. Popov, P. Gebhardt, K. Chen, R. Oldja, H. Lee, S. Murray, R. Bhargava, and N. Smolyanskiy, "Nvradarnet: Real-time radar obstacle and free space detection for autonomous driving," *arXiv preprint arXiv:2209.14499*, 2022.

[53] NVIDIA, "Duckiebot (db-j)," https://get.duckietown.com/products/duckiebot-db21, 2022.

[54] ——, "Sparkfun jetbot ai kit," https://www.sparkfun.com/products/18486, 2022.

[55] ——, "Waveshare jetbot ai kit," https://www.amazon.com/Waveshare-JetBot-AI-Kit-Accessories/dp/B07V8JL4TF/, 2022.

[56] Y. Li, Z. Li, Y. Gao, and C. Liu, "White-box multi-objective adversarial attack on dialogue generation," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 1778–1792. [Online]. Available: https://aclanthology.org/2023.acl-long.100

[57] L. Zhou, J. Gao, D. Li, and H.-Y. Shum, "The design and implementation of XiaoIce, an empathetic social chatbot," *Computational Linguistics*, vol. 46, no. 1, pp. 53–93, 2020.

[58] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 3156–3164.

[59] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 67–79.

[60] H. Zhou, S. Bateni, and C. Liu, "Sˆ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 190–201.

[61] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 107–118.

[62] S. Bateni and C. Liu, "Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 371–385.

[63] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 392–405.

[64] V. Nigade, P. Bauszat, H. E. Bal, and L. Wang, "Jellyfish: Timely inference serving for dynamic edge networks," in *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*. IEEE, 2022, pp. 277–290. [Online]. Available: https://doi.org/10.1109/RTSS55097.2022.00032

[65] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 235–247.

[66] J. Jiang, Z. Luo, C. Hu, Z. He, Z. Wang, S. Xia, and C. Wu, "Joint model and data adaptation for cloud inference serving," in *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*. IEEE, 2021, pp. 279–289. [Online]. Available: https://doi.org/10.1109/RTSS52674.2021.00034

[67] Z. Li, Y. Zhang, A. Ding, H. Zhou, and C. Liu, "Efficient algorithms for task mapping on heterogeneous cpu/gpu platforms for fast completion time," *Journal of Systems Architecture*, vol. 114, p. 101936, 2021.

[68] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *Advances in neural information processing systems*, vol. 32, 2019.

[69] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[70] Y. Li, X. Yu, Y. Liu, H. Chen, and C. Liu, "Uncertainty-aware bootstrap learning for joint extraction on distantly-supervised data," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 1349–1358. [Online]. Available: https://aclanthology.org/2023.acl-short.116

[71] M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D. I. Kim, X. Shen, V. C. M. Leung, and H. V. Poor, "Unleashing the power of edge-cloud generative AI in mobile networks: A survey of AIGC services," *CoRR*, vol. abs/2303.16129, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.16129

[72] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 443–462. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/gujarati

[73] Y. Chen, S. Chen, Z. Li, W. Yang, C. Liu, R. Tan, and H. Li, "Dynamic transformers provide a false sense of efficiency," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 7164–7180. [Online]. Available: https://aclanthology.org/2023.acl-long.395

[74] S. Chen, Z. Song, M. Haque, C. Liu, and W. Yang, "Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 365–15 374.

[75] M. Haque, R. Shah, S. Chen, B. Sisman, C. Liu, and W. Yang, "Slothspeech: Denial-of-service attack against speech recognition models," *CoRR*, vol. abs/2306.00794, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.00794

[76] S. Chen, C. Liu, M. Haque, Z. Song, and W. Yang, "Nmtsloth: understanding and testing efficiency degradation of neural machine translation systems," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1148–1160.

[77] S. Chen, M. Haque, C. Liu, and W. Yang, "Deepperform: An efficient approach for performance testing of resource-constrained neural networks," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–13.

[78] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[79] S. Chen, S. Wei, C. Liu, and W. Yang, "Dycl: Dynamic neural network compilation via program rewriting and graph optimization," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 614–626.