InferCept: Efficient Intercept Support for Augmented Large Language Model Inference

Reyna Abhyankar [*] [1] **Zijian He** [*] [1] **Vikranth Srivatsa** [1] **Hao Zhang** [1] **Yiying Zhang** [1]

Abstract

Large language models are increasingly integrated with external environments, tools, and agents like ChatGPT plugins to extend their capability beyond language-centric tasks. However, today's LLM inference systems are designed for standalone LLMs. They treat each external interaction as the end of LLM generation and form a new request when the interaction finishes, causing unnecessary recomputation of already computed contexts, which accounts for 37-40% of total model forwarding time. This paper presents InferCept, the first LLM inference framework targeting augmented LLMs and supporting the efficient interception of LLM generation. InferCept minimizes the GPU resource waste caused by LLM interceptions and dedicates saved memory for serving more requests. InferCept improves the overall serving throughput by $1.6\times$-$2\times$ and completes $2\times$ more requests per second compared to the state-of-the-art LLM inference systems.

1 Introduction

Large language models (LLMs) (Achiam et al., 2023; Touvron et al., 2023) have shown immense potential in various applications, such as natural language understanding and content generation. Nonetheless, LLMs alone can only generate text. To extend LLMs' capabilities to handle more diverse and open-ended tasks, a recent trend is to augment LLMs with external tools (Mialon et al., 2023; Wang et al., 2023; Qin et al., 2023) such as arithmetic calculation (Hao et al., 2023) and ChatGPT plugins (OpenAI, 2023), non-LLM models like image generation (Betker et al., 2024), and interaction with humans and other environments in real-time, such as virtual environments (Shridhar et al., 2021). Notably, Ng poses that "AI agent workflows will drive massive AI progress in 2024" (Ng, 2024). More generally, we can view the use of tools and the interaction with humans or other environments all as LLMs being augmented and *intercepted* by external entities.

Augmentations have unique properties that bring new problems to LLM inference systems. First, they *intercept* regular LLM decoding, *i.e.*, when an external tool is called or when waiting for human/environment responses, the normal decoding phase is paused and can only resume when the interception finishes. Second, an interception's time varies drastically across augmentation types and requests. For example, a calculator API finishes in less than 1 ms, yet it may take humans more than 1 minute to digest and type a subsequent chat prompt. This complicates request scheduling. Third, unlike regular requests, the context (*i.e.*, KV caches) cannot be used for a paused request during interceptions but will be needed upon the end of interceptions. It is unclear how to handle *temporarily unused context*. Adding to the complexity is the high variation in context length and interception time across requests.

Unfortunately, existing LLM inference systems (Kwon et al., 2023; Yu et al., 2022; Li et al., 2023; Aminabadi et al., 2022; NVIDIA, 2023; Agrawal et al., 2023) fall short in serving augmented LLMs with interceptions. They typically interpret an interception as a termination signal – they conclude the ongoing request when the interception happens, discard its context, relegate the task to the augmenting entity, and subsequently re-initiate a new request upon receiving the response from the entity. Consequently, when a request frequently triggers interceptions, for each interception, the initial prompt and all tokens generated before the interception are formed as a new request submitted to the inference system, causing substantial recomputation of keys and values (known as the KV cache) for all previously processed tokens. This recomputation wastes GPU resources, which could instead serve new requests.

In contrast to the aforementioned interception handling approach (which we refer to as `Discard`), an alternative approach is to preserve a request's context during the inter-

[*]Equal contribution [1]University of California, San Diego, La Jolla, United States. Correspondence to: Reyna Abhyankar <vabhyank@ucsd.edu>, Zijian He <zih015@ucsd.edu>, Yiying Zhang <yiying@ucsd.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

ception and resume the request when the interception finishes. This approach, which we call *Preserve*, avoids recomputation but occupies GPU memory during the entire interception. This GPU memory could accommodate other requests, which increases throughput (Kwon et al., 2023).

To avoid recomputation and memory preservation, another possible approach to handle interceptions is to swap contexts to CPU memory when an interception happens (we call it *Swap*). Although Swap avoids recomputation and GPU memory wastage, with limited GPU-CPU link bandwidth, foreground tasks (normal forwarding) could be bottlenecked by waiting for swapping to finish. The GPU memory could be used for processing more requests.

How should an LLM inference system efficiently support interceptions? To answer this question, we first study the computational patterns of six typical augmentations: arithmetic, question-and-answer, virtual environment, chatbot, image generation, and text-to-speech transformation. We then evaluate how well the state-of-the-art LLM inference system, vLLM (Kwon et al., 2023) (which performs Discard), and the two possible extensions (Preserve and Swap) perform on these augmentations.

Based on our findings, we propose INFERCEPT, the *first* LLM inference framework targeting augmented LLMs with interceptions. The core idea of INFERCEPT, which we call *min-waste interception*, is to minimize GPU memory waste caused by interceptions so that the same amount of GPU resources can be used to serve more requests. To this end, INFERCEPT incorporates three contributions. First, we deduce three waste-calculation equations to quantify the GPU memory waste of Discard, Preserve, and Swap.

Second, we improve individual interception techniques to reduce or eliminate their memory waste. For Swap, we propose swap pipelining, which overlaps swap and foreground computation in a model-layer-by-layer manner. Additionally, we split the swapping of a sequence into multiple model-forwarding iterations so that each iteration's swapping needs stays within what the GPU-CPU link can sustain (*i.e.*, a swap budget). As a result, INFERCEPT completely eliminates GPU memory waste of Swap. For Discard, we observe that certain GPU processing capacity is unused by running requests at each iteration because of decoding's imbalanced memory/compute requirements. We utilize this unused capacity for recomputation. We split requests' contexts into chunks, each small enough to fit in the unused capacity and recomputed in one iteration.

Finally, INFERCEPT dynamically chooses the interception and resumption strategies within and across requests to minimize overall GPU memory waste while ensuring fairness. For interception scheduling, we sort intercepted requests by their potential memory waste and assign the swap budget to requests with the highest potential waste. We discard or preserve the remaining requests' context based on the waste comparison. For resumption scheduling, we assign the swap budget to swapped-out requests with the earliest original arrival times. We also schedule discarded requests, preserved requests, and non-intercepted waiting requests according to FCFS (first-come first-served), adding requests until the GPU's processing capacity is reached.

We implement INFERCEPT on top of vLLM (Kwon et al., 2023), a state-of-the-art LLM inference system. We evaluate INFERCEPT, vLLM (Discard), Preserve, and Swap on A100 GPUs using three LLMs (GPT-J-6B (Wang & Komatsuzaki, 2021), Vicuña-13B (Zheng et al., 2023a), and Llama3-70B (Meta, 2024)) and the six interception types we study. Overall, INFERCEPT sustains 1.6×-2× higher serving load than vLLM while maintaining similar latency per token generation. INFERCEPT also achieves over 2× more completed requests per second.

INFERCEPT is available at https://github.com/WukLab/InferCept.

2 Augmented LLMs

This section first discusses popular augmented LLM frameworks and then presents properties of typical augments from our empirical study.

2.1 Augemented LLM Frameworks

To extend LLMs' ability to undertake more types of tasks, different approaches have been proposed to *augment* an LLM with various external entities (Mialon et al., 2023). Below, we describe typical augmentations in three categories.

The first type of augmentation involves non-LLM tools called during the decoding phase of an LLM, usually aimed at extending the types of tasks the LLM can handle. These tools range from simple tools like calculator (Wolfram, 2023) and calendar to real-world interactions like information retrieval (Baeza-Yates et al., 1999) and restaurant reservation (OpenTable, 2023). A tool augmentation can also be another machine-learning model such as translation (Costa-jussà et al., 2022), and question-answering (QA) (Izacard et al., 2022). Previous research (Parisi et al., 2022; Hao et al., 2023; Schick et al., 2023) has shown the effectiveness of fine-tuning an LLM to generate appropriate tool calls automatically during the decoding phase. Instead of fine-tuning, another approach is instructing LLMs to select tools or models for a user task (Shen et al., 2023; Lu et al., 2023).

The second type augments a single triggering of an LLM by making a *sequence* of calls to the same LLM. For exam-

Type	Int Time (sec)	Num Interceptions	Context Len
Math	(9e-5, 6e-5)	(3.75, 1.3)	(1422, 738)
QA	(0.69, 0.17)	(2.52, 1.73)	(1846, 428)
VE	(0.09, 0.014)	(28.18, 15.2)	(2185, 115)
Chatbot	(28.6,15.6)*	(4.45,1.96)	(753, 703)
Image	(20.03, 7.8)†	(6.91,3.93)*	(1247,792)
TTS	(17.24,7.6)†	(6.91,3.93)*	(1251,792)

Table 1: **Interception Properties** *Each cell shows (mean, variance) of the augment. * estimated behavior. † partially estimated.*

ple, to continuously interact with a human, a chatbot can take rounds of back-and-forth chats by calling an LLM at each step and maintaining a chat history (Greyling, 2023). Another popular use case is decomposing a complex task into multiple steps. For example, Chain-of-Thought (Wei et al., 2023; Zhang et al., 2023) breaks one request into a chain of "thought" steps, each being a new prompt to an LLM that continues the history of thoughts. Similarly, ReAct (Yao et al., 2023) uses a chain of reasoning and action steps to accomplish complex tasks.

The third type allows for more complex compositions of LLMs, other models, and tools. Frameworks such as LangChain (Chase, 2022), DSpy (Khattab et al., 2024), Gorilla (Patil et al., 2023), SGLang (Zheng et al., 2023b), and AgentGraph (Chen et al., 2019) provide programming models for users to write their own flows of augmented LLMs. Other efforts enable LLMs to generate compositions of augmented LLMs (Suris et al., 2023; Qian et al., 2023).

2.2 Augmenting Entities and Their Properties

To understand augmented LLM interceptions and design inference systems for them, we first examine a set of representative augmentations, including arithmetic (math), question-and-answer (QA), virtual environment (VE), chatbot, image generation, and text-to-speech (TTS), to answer three key questions: 1) how long does an interception take? 2) how many interceptions occur for a request? and 3) what is the context length when an interception happens? Table 1 summarizes our results with averages and variations. We leave other metrics, such as interception returned token length, detailed CDF distributions, and our evaluation methodology, in the Appendix.

Arithmetic (Math). To solve complex math problems, an LLM can be augmented to deconstruct the problem and call a calculator tool step-by-step. We evaluate this use case with the GSM8K-XL (Hao et al., 2023) dataset, which contains 8.5K high-quality grade-school math problems. As expected, the calculator's execution times are short, an average of 0.2 ms. It has a fairly large context length when calculators are called since we must prompt the LLM with demonstrations of solving problems step-by-step.

Knowledge-based question and answer (QA). To allow pre-trained LLMs to access wider sets of knowledge, one can augment the LLM to call knowledge-based QA tools that retrieve information from a rich dataset. We use the Multihop QA Wikipedia (Yang et al., 2018) dataset to evaluate this use case. While having longer context lengths for questions, these QA tools provide relatively quick yet variable execution times due to the non-deterministic network communication latency with Wikipedia.

Virtual Environment (VE). An LLM can be augmented to interact with virtual environments and accomplish complex tasks by performing gradual steps (Gu et al., 2022; Huang et al., 2022; Hu et al., 2023). To evaluate VE, we use the ALFWorld dataset (Shridhar et al., 2021), an interactive environment that aligns text descriptions and commands with a physically embodied robotic simulation. The VE interaction has a short and stable interception time due to a locally executed and embodied text-based environment. The context length is long due to the instructions needed to prompt the LLM to understand VE action sequences.

Chatbot. A popular task for LLMs is a chatbot, which interacts with humans with a sequence of chats. When a human receives a chat response from the LLM, the LLM generation sequence is essentially intercepted. The LLM generation is resumed when the human sends the subsequent chat message. During this process, the chat history must be kept as the context for the entire chat sequence. We use the ShareGPT dataset (Zheng et al., 2023a), which contains crowd-sourced real-user ChatGPT conversations to evaluate chatbots. We estimate human response time (*i.e.*, the interception time) as the summation of human scanning time (Brysbaert, 2019) of the previous chat response and typing time (Ma et al., 2015) of the next prompt. The context lengths vary greatly due to the variety of human prompts. Its interception time also has a high variance.

Image generation. An LLM can be augmented to generate an image or a sequence of images when a human gradually refines features through multiple prompts (*e.g.*, adding details to a face depiction). This involves two interceptions: one for triggering an image-generation model and another for receiving user subsequent prompts. To understand this use case, we use ChatGPT to create a dataset by generating a series of image-generation prompts, each triggering a call to the Stable Diffusion model (Rombach et al., 2021). This workload's interception time has high variation due to variations in executing the diffusion model and the variation in human response time. The average context length is shorter than the chatbot interception due to smaller input prompts to the diffusion model.

Text-To-Speech (TTS). As demonstrated by the ChatGPT Whisper API support (Brockman et al., 2023), TTS can be integrated with an LLM to communicate with a human. Similar to our image-generation dataset, we use Chat-

GPT to generate a series of prompts, each triggering a call to the Bark TTS model (AI, 2023). Similar to image generation, we estimate the human response time but measure actual TTS execution time. TTS's behavior also has high variation in interception time due to variations in model execution time and user response time.

Summary and insights. Overall, we find LLM interception time is highly dependent on augmentation types and exhibits a clear difference between short-running (Math, QA, VE) and long-running (Chatbot, Image, TTS) ones. The short-running augmentations are fully automated with no human interaction. Most also exhibit small variations (except for QA, which interacts with the network). Augmentations that involve human interaction and/or call another large model are long-running with high variations. The short/long interception times and their variation imply that no single LLM interception strategy is universally applicable, but an inference system could utilize augmentation type as a hint for interception time. Context lengths for all the augmentations are large, implying potentially high GPU memory wastage. Context lengths also see high variation, complicating the handling of them at runtime.

3 Existing LLM Inference Systems

This section discusses related work in LLM inference systems and why simple extensions to them are insufficient for interception handling, as shown in Figure 1.

3.1 LLM Inference Systems

Recent years have seen the rise of inference systems that are designed specifically for LLMs. Unlike INFERCEPT, none of the existing inference systems are designed for handling interceptions, and all suffer from unoptimized performance, as we will show in Section 3.2.

Some works aim to improve overall inference throughput with better scheduling strategies. For example, Orca (Yu et al., 2022) proposes iteration-level scheduling where, at the end of each model forward pass, Orca's scheduler is invoked to form a new batch for the next forward pass. Such iteration-level scheduling strategies improve GPU utilization, resulting in higher inference throughput. INFERCEPT also adopts iteration-level scheduling but with a unique and new scheduling strategy — at each iteration, INFERCEPT makes a decision that minimizes GPU memory wastage for requests intercepted or resumed in that iteration based on their interception properties. Besides FCFS, which Orca and others use, alternative request scheduling policies such as a multi-level feedback queue in FastServe (Wu et al., 2023) have been proposed. We leave the comparison of these scheduling policies to future work.

Another optimization target is GPU memory usage. vLLM (Kwon et al., 2023) proposes the concept of paged attention, where the KV cache is treated as virtual memory that can map to non-contiguous physical GPU memory. Because of the flexibility in virtual memory, vLLM achieves better utilization of GPU memory, thereby improving overall inference performance. INFERCEPT also utilizes paged attention to efficiently use GPU physical memory. Unlike vLLM, which is agnostic to interceptions, INFERCEPT adaptively preserves, discards, or swaps the KV cache of intercepted requests to minimize memory waste. Other memory-efficient techniques, such as prefix sharing in SGLang (Zheng et al., 2023b) and Preble (Srivatsa et al., 2024), are orthogonal and can be added to INFERCEPT.

Yet another set of LLM inference systems focuses on the imbalanced computation needs between the prefill and decoding stages. Sarathi (Agrawal et al., 2023) is a system that proposes the technique of chunked prefill, which splits prompt tokens into chunks, each merged with other decoding requests to form one iteration's batch. INFERCEPT's chunking splits sequences for recomputation and swapping to fully utilize GPU and GPU-CPU-link resources while minimizing memory waste. Other proposals on prefill-decoding optimization like DistServe (Zhong et al., 2024) are orthogonal and can potentially be added to INFERCEPT.

3.2 Limitations of Inference Systems and Extensions

We now qualitatively and quantitatively analyze the limitations of existing LLM inference systems and naive extensions to them to support interceptions. We quantify the impact of these approaches on serving speed with our proposed *GPU memory waste* calculations.

Discard-based approaches. Today's LLM inference systems are not designed for augmented LLMs. To serve LLMs with augmentations, such a system would need to treat the interception as the end of the request and re-initiate a request when the interception finishes. As today's inference systems use the FCFS scheduling policy, they schedule these re-initiated requests at the end of the waiting queue and starve resumed requests. An easy fix to this scheduling problem is using an intercepted request's original arrival time when re-inserting it into the waiting queue (we call this scheme `ImprovedDiscard`).

`Discard` and `ImprovedDiscard` both require the recomputation of all context tokens and incur GPU memory waste from two sources. First, recomputation consumes memory that is not used to produce any new tokens. Assuming the context of request i when interception j occurs has C_i^j tokens, each token's KV cache occupies M memory, and recomputation takes $T_{fwd}(C_i^j)$ time, where T_{fwd} is a mapping from the number of scheduled tokens in a batch to the execution time of that iteration.

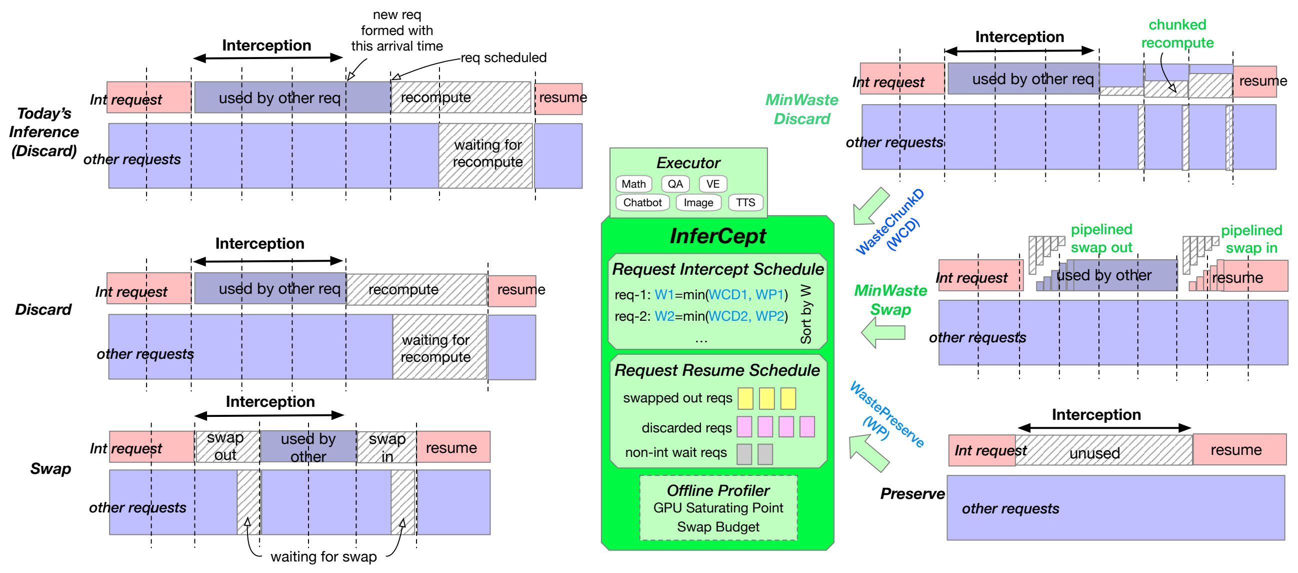

Figure 1: **INFERCEPT and Alternative Approaches.** *Vertical: GPU resources occupied or wasted by intercepted requests and other normal ones. Horizontal: timeline divided by iterations (dotted vertical lines). Hatch parts represent memory waste.* INFERCEPT *adaptively combines preserve, our optimized discard, and our optimized swap. All green parts represent* INFERCEPT*'s techniques.*

The memory wasted for recomputation is $T_{fwd}(C_i^j) \times C_i^j \times M$. Second, recomputing the context increases iteration time, as an iteration now needs to finish all the recomputations alongside model forwarding of other running requests. The memory occupied by the other running requests is wasted during the additional iteration time, $T_{fwd}(C_i^j)$. Thus, the memory waste for this reason is $T_{fwd}(C_i^j) \times C_{other} \times M$, where C_{other} is the sum of the contexts of other requests. The total memory waste for `Discard` and `ImprovedDiscard` is:

$$\begin{aligned}\text{WasteDiscard}_i^j = T_{fwd}(C_i^j) \times C_i^j \times M \\ + T_{fwd}(C_i^j) \times C_{other} \times M\end{aligned} \quad (1)$$

In practice, our evaluation shows that `Discard` incurs a GPU resource wastage (GB*min) of 27%, and 37-40% of the total model forwarding time is spent on recomputation with a mixed workload containing all six augmentations in §2.

Preserve-based approach. Instead of discarding, one could preserve the context of a request when interception happens. This `Preserve` strategy avoids recomputation cost and can immediately resume a request when the interception finishes. However, the preserved context wastes GPU memory while the request remains paused. The preserve waste for request i when interception j occurs is the duration of that interception, T_{INT}^j, multiplied by the amount of GPU memory held by the request's context.

$$\text{WastePreserve}_i^j = T_{INT}^j \times C_i \times M \quad (2)$$

Our measured GPU memory waste with real interceptions is surprisingly high: nearly half of the GPU memory is occupied by interrupted requests more than 60% of the time.

Swap-based approach. To avoid recomputation overhead in `Discard` and memory waste in `Preserve`, one potential technique is to swap all context data from GPU to CPU memory when an interception happens and swap it back to GPU when the interception ends. The straightforward implementation of `Swap` performs swapping in a synchronous manner by launching CUDA kernels for moving data out/in and letting computation utilize the relevant memory space. Swapping can take an extremely long time with huge contexts because of the volume of data to be moved and the kernels to be launched. To elaborate on the latter, multiple kernels need to be launched, each for a discontinuous physical memory region. With PagedAttention (Kwon et al., 2023), the context of an intercepted request can scatter across many physical memory regions, causing high kernel launch overhead.

Like `Discard`, `Swap` incurs memory waste from two sources. First, the memory space being swapped is wasted during swapping, accounting for $T_{swap}(C_i^j) \times C_i^j \times M$, where T_{swap} is a mapping from the number of tokens to swap to the corresponding swapping latency. Second, `Swap` could increase the iteration time, causing all running requests to wait for swapping to finish and do nothing. This waste is $T_{swap}(C_i^j) \times C_{other} \times M$. The total waste doubles to account for swapping in and out:

$$\text{WasteSwap}_i^j = 2 \times T_{swap}(C_i^j) \times C_{batch} \times M \quad (3)$$

Our evaluation shows that `Swap` wastes 26% GPU resources, and over 25% of the total workload time is spent on waiting for swapping for the mixed workload.

4 INFERCEPT

This section presents INFERCEPT, first with our improved swap and recomputation mechanisms, then with our intercepting and resuming request scheduling, and ending with interception duration estimation and implementation details. Figure 1 shows INFERCEPT's overall architecture.

4.1 Swap Pipelining and Chunking

We discuss how INFERCEPT avoids swap memory waste.

Swap pipelining and overlapping. `Swap` performs swapping in a serial manner and blocks foreground computation for the entire swapping time. To mitigate this problem, we propose to perform swapping in a pipelined manner and in the background. Specifically, INFERCEPT views each model layer's swapping as one pipeline stage and pipelines kernel launching, data movement, and foreground model forwarding. For example, when we launch the swap kernel for layer $i+2$, we move the context for layer $i+1$, and layer i's context has been freed and used for normal forwarding.

Swap chunking. To further reduce memory waste, we propose to chunk swap-out and swap-in across multiple iterations so that in each iteration, the swap latency can be hidden by overlapping it with model forwarding. We obtain T_{fwd} through offline profiling and calculate T_{swap} based on the swapping bandwidth and the per-token memory requirements M. At iteration i with batch size B_i, we set $T_{swap}(N_i) = T_{fwd}(B_i)$ and calculate N_i. N_i, what we call the *swap limit*, indicates the number of tokens that can be swapped for free (*i.e.*, hidden behind model forwarding).

Determining swap-in and swap-out budget. At every iteration, there can be tokens waiting to be swapped out and swapped in. INFERCEPT determines how much bandwidth to give to swap-out and to swap-in at each iteration by maximizing inference throughput (*i.e.*, the number of tokens that can be added to an iteration's processing) while guaranteeing the following criteria: 1) the total amount of swap-in and swap-out tokens should be at most N_i; 2) the amount of swapped-out memory should not exceed the amount of free CPU memory plus the swapped-in memory; and 3) the amount of swapped-in memory and memory for newly scheduled tokens (*i.e.*, our maximizing target) should not exceed the amount of swapped-out memory and free GPU memory. We use the obtained swap-in and swap-out budget when deciding the request interception schedule in §4.3.

4.2 Recomputation Chunking

Unlike `Swap`, which consumes GPU-CPU-link bandwidth and can be hidden from foreground GPU tasks, the recomputation of discarded tokens requires GPU resources that cannot be avoided. However, we observe that decoding and recomputation have complementary resource requirements, with decoding requiring less GPU core resource (for one query token) than recomputation (queries for the entire context length) for the same amount of memory. Thus, a batch of decoding requests usually cannot fill all GPU cores before running out of GPU memory. Mixing decoding and recomputation requests can increase GPU core utilization while staying within its memory boundary. The main challenge is how much recomputation to mix with decoding.

`Discard` and `ImprovedDiscard` impose the recomputation of an entire request in one iteration. Recomputation of a long context would add more computation burden than what a GPU can handle, causing significantly increased iteration time. As a result, long $T_{fwd}(C_i^j)$ causes huge $WasteDiscard$ in Equation 1.

To mitigate this issue, we propose an adaptive technique that separates the recomputation of a context sequence into chunks, each added to one iteration without going beyond what the GPU can sustain. We observe that the number of query tokens that all GPU cores can process in parallel is limited and fixed for a given model architecture. We call this number the GPU *saturation point*, S, expressed as query token count. Processing more query tokens beyond S increases iteration time without improving serving throughput. INFERCEPT obtains S from offline profiling and sets the chunk size as S minus the running group size.

The memory waste of our chunked recomputation mechanism can be calculated in two parts, similar to Equation 1. For the recomputation itself, we add one chunk of memory waste per iteration that lasts until all chunks have been recomputed, which essentially cuts the waste of `Discard`'s all-recomputation-at-once scheme by half, *i.e.*, $T_{fwd}(C_i^j) \times C_i^j \times M/2$, as illustrated in Figure 1. The second part of waste comes from other requests' occupied memory during recomputation-added iteration time. This waste amounts to $n \times T_{fwd}(\frac{C_i^j}{n}) \times C_{other} \times M$, where n is the number of iterations to finish recomputing the request and $T_{fwd}(\frac{C_i^j}{n})$ is the per-chunk added iteration time. Adding these two parts, we have the total recomputation memory waste of chunked recomputation as

$$\begin{aligned}\text{WasteChunkD}_i^j = {} & \frac{T_{fwd}(C_i^j) \times C_i^j \times M}{2} \\ & + n \times T_{fwd}(\frac{C_i^j}{n}) \times C_{other} \times M\end{aligned} \quad (4)$$

Compared to Equation 1, the left term (recomputation itself) cuts the corresponding term of `Discard` by half, and the right term (other requests) is no larger than the right term of `Discard` because $n \times T_{fwd}(\frac{C_i^j}{n}) \leq T_{fwd}(C_i^j)$. Meanwhile, INFERCEPT increases GPU core utilization and further improves overall serving throughput.

4.3 Inter-Request Action Decision

Scheduling intercepted requests. We now discuss how INFERCEPT schedules intercepted requests using a hybrid of preserve, our improved swap, and our improved discard when considering multiple requests together. First, for each request i encountering interception j, we calculate its memory waste as the minimum of WastePreserve (Equation 2) and WasteChunkDiscard (Equation 4):

$$\text{Waste}_i^j = \mathbf{min}(\text{WastePreserve}_i^j, \text{WasteChunkD}_i^j) \tag{5}$$

Next, we sort all intercepted requests in descending order based on their memory waste. We swap out context from these requests according to this order until we run out of the swap-out budget. For the remaining paused requests, we preserve or discard their remaining context based on its decision from Equation 5.

Scheduling resumed and other waiting requests. At the beginning of each iteration, INFERCEPT determines what requests to insert into the batch for the iteration. To facilitate this decision, INFERCEPT maintains three queues: a running queue containing currently running requests in the system, a swap queue containing requests that have been resumed but are previously swapped out during the interception, and a waiting queue. The waiting queue contains discarded requests that have been resumed, new requests received by INFERCEPT that have never been served, and previously running requests that have been evicted due to the lack of GPU resources. The latter two types are the same as today's inference systems. Each queue is sorted by its requests' original arrival times.

At each iteration, we choose requests from the waiting queue in the FCFS order until the GPU saturation point is reached; FCFS ensures fairness and avoids starvation. Requests chosen can be any of the three aforementioned waiting types. If it is a discarded request, the iteration will perform the recomputation of the scheduled number of tokens. If this request has only been partially recomputed, it remains in the wait queue with the remaining tokens. At each iteration, we also choose requests from the swap queue in the FCFS order until the swap-in budget is reached. We maintain and schedule a separate swap queue because the swap-in budget is additional to GPU resources and should always be utilized by resumed requests as much as the budget allows.

4.4 Interception Duration Estimation

The calculation of memory waste for preserving context (Equation 2) requires interception duration. For interceptions with highly variable durations or those not profiled offline, we propose a dynamic estimation method by setting $\hat{T}_{INT} = t_{now} - t_{call}$ where t_{now} is the current time updated each iteration and t_{call} is when the last interception was initiated. Effectively, the longer a request is intercepted, the larger the estimated interception time. Our evaluation shows that INFERCEPT using this estimation method achieves 93% of the performance compared with using an oracle providing exact interception durations in mixed workload.

4.5 Implementation

INFERCEPT comprises four key components: a scheduling policy, waste calculation, chunked recomputation and swapping, and diverse augmentation support. We implement INFERCEPT on top of vLLM (Kwon et al., 2023) to leverage its PagedAttention technique for regular LLM memory management, Most of our techniques are highly modular and orthogonal to optimizations designed for non-intercepted LLMs. Thus, INFERCEPT can be potentially integrated into other LLM serving systems like DeepSpeed (Aminabadi et al., 2022), Orca (Yu et al., 2022), and TensorRT-LLM (Vaidya et al., 2023).

5 Evaluation Results

We compare INFERCEPT to vLLM (with no modifications), `ImprovedDiscard`, `Preserve`, and `Swap`, as discussed in §3.2. To evaluate INFERCEPT and the baselines, we augment the 6B-parameter GPT-J model (Wang & Komatsuzaki, 2021), the 13B Vicuna model (Zheng et al., 2023a), and the 70B Llama3 model (Meta, 2024) with workloads presented in §2. We run augmented GPT-J on one NVIDIA A100 GPU. For Vicuna, we use two environments: running on a single A100 GPU and distributed on two A100 GPUs with tensor parallelism. For Llama3, we distribute it on four A100 GPUs with tensor parallelism. To mimic real-world serving scenarios that often receive different types of requests, we use a request dataset that merges the six augmentations presented in §2 by uniformly sampling requests from them.

5.1 End-to-End Performance

We first compare the end-to-end performance of INFERCEPT and the baselines. Following recent LLM inference research papers (Kwon et al., 2023; Yu et al., 2022), we first report the serving throughput as normalized latency (*i.e.*, the median of every request's end-to-end latency divided by its output length) when varying request load (number of

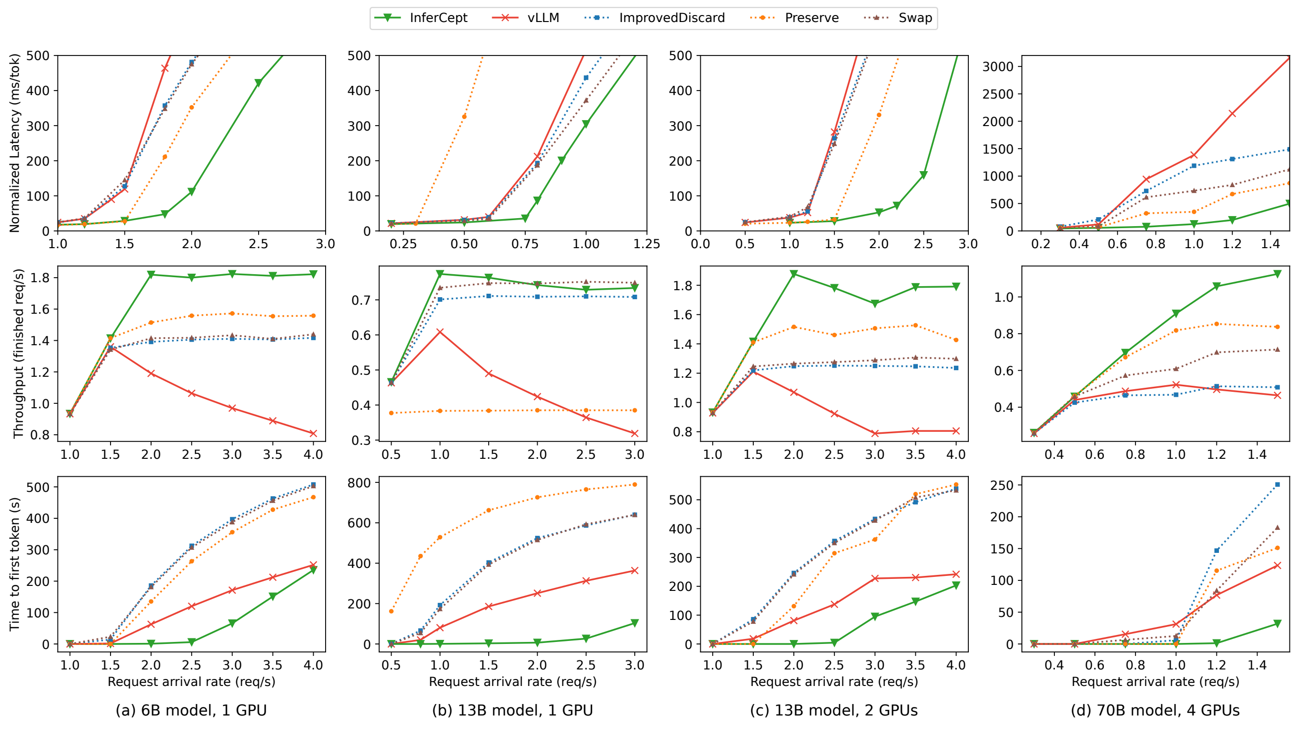

Figure 2: **End-to-end Performance on Mixed Workload.** ***First Row: Normalized Latency.*** *Lower right is better, i.e., sustains higher serving load.* ***Second Row: Throughput.*** *Expressed as completed requests per second. Higher is better.* ***Third Row: Time-to-first-token (TTFT).*** *Lower is better, i.e., shorter response time.*

requests arrived per second). We also remove a request's intercepted time from its end-to-end latency because it is the same across all serving systems. Under the same normalized latency, a higher-throughput system should sustain a higher request rate. Apart from normalized latency, we also report the number of finished requests per second (*i.e.*, throughput) and the time from request arrival to the first generated token (*i.e.*, TTFT). These metrics are important in meeting service-level objectives and system response time (Zhong et al., 2024; Hu et al., 2024; Patel et al., 2024). Figure 2 shows these results for INFERCEPT and the baselines on all three models. For Vicuna-13B, we report both a single GPU and two GPU executions.

6B model results. INFERCEPT outperforms baselines across all metrics. INFERCEPT sustains up to 1.6× higher request arrival rate at the same normalized latency compared to vLLM. Under the same request rate, INFERCEPT's normalized latency is 1.9×-5.7× lower than vLLM. INFERCEPT serves 1.7× higher request arrival rate than vLLM with the smallest TTFT. INFERCEPT outperforms all of the baselines mainly because of our min-waste-based schedule. Moreover, INFERCEPT improves the recomputation and swap mechanisms over `ImprovedDiscard` and `Swap` with chunking and pipelining. Compared to them, INFERCEPT eliminates over 60% of GPU waste caused by recomputation and 96% of GPU waste for swapping.

The vanilla vLLM suffers from high recomputation waste and delays in scheduling resumed requests per FCFS policy, resulting in worse throughput and normalized request latency. Its TTFT is better than other baselines and close to INFERCEPT because it puts intercepted requests to the end of the wait queue, which is a round-robin schedule. This results in the system primarily handling requests before their first interception, essentially allowing TTFT not to be affected by interceptions. `ImprovedDiscard` maintains an intercepted request's original arrival time. Thus, both its normalized latency and throughput are better than vanilla vLLM. However, `ImprovedDiscard` still incurs the same recomputation overhead and underperforms INFERCEPT on all metrics. `Preserve` is the best among all the baselines regarding normalized latency and throughput, showing that the preserving overhead for our mixed workload with small models is lower than the recomputation overhead for most requests. Yet, `Preserve` is worse than INFERCEPT on all metrics because of INFERCEPT's dynamic min-waste scheduling and improved swapping and recomputation. The overall results for `Swap` are similar to `ImprovedDiscard`. Both baselines avoid GPU memory waste during interceptions. While `ImprovedDiscard` causes running requests to wait for the recomputation requests in an iteration, `Swap` causes running requests to wait for swapping out/in requests in an iteration. These overheads have similar effects on the final performance metrics.

13B model results. When running the 13B model on a single GPU, INFERCEPT outperforms all the baselines in normalized latency but with smaller improvements than the 6B model. A larger model's weights occupy more GPU memory, reducing space for the KV cache. Moreover, the increased normal forwarding time contributes to most of the normalized latency, limiting the room for improvement. Still, we serve up to 1.25× higher request rates without a noticeable increase in the normalized latency and also sustain 3.1× higher load with the smallest TTFT.

A more practical scenario for large model inference is a distributed execution environment. We evaluate the 13B model with tensor-parallelism (Shoeybi et al., 2019) on two GPUs. INFERCEPT outperforms all baselines by a larger margin than when running on a single GPU. INFERCEPT serves up to 1.8× higher request arrival rates and a 1.6×-10× improvement in normalized latency compared to vLLM. We obtain more benefits in the distributed setting because each GPU uses less memory for storing model weights and allocates more space for the KV cache. This allows more requests to be served concurrently and thus introduces more interceptions. As INFERCEPT minimizes GPU memory waste from these interceptions, the improvement over baselines becomes more pronounced in this setting.

70B model results. For the Llama3-70B model, INFERCEPT has substantial improvement over all baselines. INFERCEPT can sustain 2× higher request arrival rate and achieve 1.3×-12× lower normalized latency at the same RPS. We also achieve 2.4× higher load than vLLM with the same TTFT. As model size increases, both vanilla vLLM and `ImprovedDiscard` experience significant recomputation costs. In contrast, `Preserve` and `Swap` perform better. This is attributed to the adoption of grouped query attention (GQA) (Ainslie et al., 2023), which compresses the attention KVs. This mechanism reduces inactive memory for `Preserve` and moderates data movement for `Swap`. By making optimal preemption decisions tailored to specific requests, INFERCEPT provides more benefits since the GPU waste is lowered significantly.

Single-augment workloads. Apart from the mixed workload, we evaluate INFERCEPT on two single-API workloads: QA and Chatbot (§2). INFERCEPT outperforms vLLM by up to 2.3× and 1.9× on normalized latency with QA and Chatbot respectively. INFERCEPT's improvement with QA is larger because most QA API calls are short and have smaller waste when using preserve than discard.

5.2 Performance Deep Dive

To further understand INFERCEPT's benefits, we break down its techniques by adding them one at a time to vanilla vLLM. Using the mixed workload, we report the normalized latency and GPU memory waste under the load of 2 requests per second for the 6B model in Figure 3. Other request loads have similar trends.

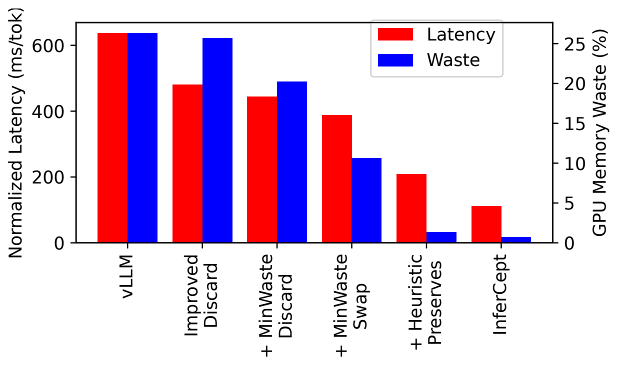

Figure 3: **INFERCEPT Technique Breakdown.** *Each bar group adds one technique over its left bar group, with the leftmost being vanilla vLLM and the rightmost being the full* INFERCEPT.

We first improve `Discard` (vanilla vLLM) by maintaining the request's original arrival time as in `ImprovedDiscard`, which reduces normalized latency by 24.5%. We then add INFERCEPT's recomputation chunking, resulting in 7.8% more improvement. Next, we add INFERCEPT's budgeted swapping (discarding once the limit is reached). This results in an added improvement of 12.7%. Then, we add preserve and use a simple heuristic to decide between discard and preserve: discard for interactive (or long-running) interceptions and preserve for automated (short-running) interceptions. This addition has a 46.1% improvement, mainly because of the added support for `Preserve`. Finally, adding min-waste-based adaptive schedule (*i.e.*, the whole INFERCEPT) results in an additional 46.4% improvement. Each of INFERCEPT's techniques reduces GPU memory waste, with the whole INFERCEPT only having 0.69% waste. This demonstrates the effectiveness of our min-waste preemption principle.

Note that INFERCEPT's techniques' benefits differ for different workloads. For example, our chunking and pipelining techniques for recomputation and swapping contribute 54% of total speedup for the single Chatbot workload. This is because Chatbot has a long interception duration, allowing for more requests to execute and, in turn, triggering more swap and recomputation.

6 Conclusion

We introduced INFERCEPT, an LLM inference framework optimized for interceptions during inference. By minimizing GPU memory waste as a unified objective, INFERCEPT delivers 1.6×-2× higher request arrival rates and achieves 1.3×-12× lower normalized latency compared to state-of-the-art inference systems.

Acknowledgements

We would like to thank the anonymous reviewers for their tremendous feedback and comments, which have substantially improved the content and presentation of this paper. We are also thankful to Longfei Yun, Shravan Konduru, Hao-Ting Tso, Chujia Guo, Yu-shu Chen, Chace Zhu, Shivam Pansuria, and Katherine Guo for their contribution to the initial phase of this work. This material is based upon work supported by gifts from AWS, Google, and Meta. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these institutions.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

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

Agrawal, A., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., and Ramjee, R. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, August 2023.

AI, S. Bark: Text-to-speech model. https://github.com/suno-ai/bark, 2023.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, Singapore, December 2023.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, Texas, November 2022. IEEE.

Baeza-Yates, R., Ribeiro-Neto, B., et al. *Modern Information Retrieval*, volume 463. ACM Press, New York, 1999.

Betker, J., Goh, G., Jing, L., Brooks, T., Wang, J., Li, L., Ouyang, L., Zhuang, J., Lee, J., Guo, Y., Manassra, W., Dhariwal, P., Chu, C., Jiao, Y., and Ramesh, A. Improving image generation with better captions. https://cdn.openai.com/papers/dall-e-3.pdf, 2024.

Brockman, G., Eleti, A., Georges, E., Jang, J., Kilpatrick, L., Lim, R., Miller, L., and Pokrass, M. Introducing chatgpt and whisper apis. https://openai.com/blog/introducing-chatgpt-and-whisper-apis, March 1 2023.

Brysbaert, M. How many words do we read per minute? a review and meta-analysis of reading rate. *Journal of memory and language*, 109:104047, 2019.

Chase, H. LangChain. https://github.com/langchain-ai/langchain, October 2022.

Chen, L., Chen, Z., Tan, B., Long, S., Gasic, M., and Yu, K. Agentgraph: Towards universal dialogue management with structured deep reinforcement learning. *arXiv preprint arXiv:1905.11259*, May 2019.

Costa-jussà, M. R., Cross, J., Çelebi, O., Elbayad, M., Heafield, K., Heffernan, K., Kalbassi, E., Lam, J., Licht, D., Maillard, J., Sun, A., Wang, S., Wenzek, G., Youngblood, A., Akula, B., Barrault, L., Gonzalez, G. M., Hansanti, P., Hoffman, J., Jarrett, S., Sadagopan, K. R., Rowe, D., Spruit, S., Tran, C., Andrews, P., Ayan, N. F., Bhosale, S., Edunov, S., Fan, A., Gao, C., Goswami, V., Guzmán, F., Koehn, P., Mourachko, A., Ropers, C., Saleem, S., Schwenk, H., and Wang, J. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*, July 2022.

Greyling, C. When using the chatgpt api, users will have to manage the context. https://cobusgreyling.medium.com/when-using-the-chatgpt-api-users-will-have-to-manage-the-context-ba5869238913, March 6 2023.

Gu, J., Stefani, E., Wu, Q., Thomason, J., and Wang, X. Vision-and-language navigation: A survey of tasks, methods, and future directions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.524.

Hao, S., Liu, T., Wang, Z., and Hu, Z. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. In *Advances in Neural Information Processing Systems 36*, New Orleans, Louisiana, December 2023.

Hu, C., Huang, H., Xu, L., Chen, X., Xu, J., Chen, S., Feng, H., Wang, C., Wang, S., Bao, Y., Sun, N., and Shan, Y. Inference without interference: Disaggregate llm inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, January 2024.

Hu, Y., Lin, F., Zhang, T., Yi, L., and Gao, Y. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*, November 2023.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of 39th International Conference on Machine Learning*, Honolulu, Hawai'i, 2022.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. Atlas: Few-shot learning with retrieval augmented language models, 2022.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., A, S. V., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations (ICLR '24)*, Vienna, Austria, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, Koblenz, Germany, October 2023.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., and Stoica, I. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, Boston, MA, July 2023.

Lu, P., Peng, B., Cheng, H., Galley, M., Chang, K.-W., Wu, Y. N., Zhu, S.-C., and Gao, J. Chameleon: Plug-and-play compositional reasoning with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS '23)*, New Orleans, Louisiana, December 2023.

Ma, Z., Edge, D., Findlater, L., and Tan, H. Z. Haptic keyclick feedback improves typing speed and reduces typing errors on a flat keyboard. In *2015 IEEE World Haptics Conference (WHC)*, Evanston, Illinois, 2015. IEEE.

Meta. Meta llama 3. https://llama.meta.com/llama3/, 2024.

Mialon, G., Dessi, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Roziere, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., Grave, E., LeCun, Y., and Scialom, T. Augmented language models: a survey. *Transactions on Machine Learning Research (TMLR)*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=jh7wH2AzKK. Survey Certification.

Ng, A. The batch weekly issues 241. https://www.deeplearning.ai/the-batch/issue-241/, March 2024.

NVIDIA. Fastertransformer. https://github.com/NVIDIA/FasterTransformer, 2023.

OpenAI. ChatGPT plugins. https://openai.com/blog/chatgpt-plugins, March 2023.

OpenTable. New: Chatgpt restaurant recs, powered by opentable. https://www.opentable.com/blog/chatgpt/, March 23 2023.

Parisi, A., Zhao, Y., and Fiedel, N. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, May 2022.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *The 53th International Symposium on Computer Architecture (ISCA 2024)*, Buenos Aires, Argentina, June 2024.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Qian, C., Han, C., Fung, Y., Qin, Y., Liu, Z., and Ji, H. CREATOR: Tool creation for disentangling abstract and concrete reasoning of large language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: (EMNLP '23)*, Singapore, December 2023. URL https://aclanthology.org/2023.findings-emnlp.462.

Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Huang, Y., Xiao, C., Han, C., Fung, Y. R., Su, Y., Wang, H., Qian, C., Tian, R., Zhu, K., Liang, S., Shen, X., Xu, B., Zhang, Z., Ye, Y., Li, B., Tang, Z., Yi, J., Zhu, Y., Dai, Z., Yan, L., Cong, X., Lu, Y., Zhao, W., Huang, Y., Yan, J., Han, X., Sun, X., Li, D., Phang, J., Yang, C., Wu, T., Ji, H., Liu, Z., and Sun, M. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, June 2023.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*, 2021.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and

Scialom, T. Toolformer: Language models can teach themselves to use tools. *37th Conference on Neural Information Processing Systems*, 2023.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS '23)*, New Orleans, Louisiana, December 2023.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. Alfworld: Aligning text and embodied environments for interactive learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 2021.

Srivatsa, V., He, Z., Abhyankar, R., Li, D., and Zhang, Y. Preble: Efficient distributed prompt scheduling for llm serving. *UCSD CSE Technical Reports*, May 2024. URL https://escholarship.org/uc/item/1bm0k1w0.

Suris, D., Menon, S., and Vondrick, C. Vipergpt: Visual inference via python execution for reasoning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV '23)*, pp. 11854–11864, Los Alamitos, CA, USA, October 2023. URL https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.01092.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vaidya, N., Comly, N., DeLaere, J., Patel, A., and Oh, F. Nvidia tensorrt-llm supercharges large language model inference on nvidia h100 gpus. https://developer.nvidia.com/blog/nvidia-tensorrt-llm-supercharges-large-language-model-inference-on-nvidia-h100-gpus/, 2023.

Wang, B. and Komatsuzaki, A. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Wang, L., Ma, C., Feng, X., Zhang, Z., ran Yang, H., Zhang, J., Chen, Z.-Y., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and rong Wen, J. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, August 2023. URL https://api.semanticscholar.org/CorpusID:261064713.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, New Orleans, Louisiana, 2023.

Wolfram, S. Chatgpt gets its 'wolfram superpowers'! https://writings.stephenwolfram.com/2023/03/chatgpt-gets-its-wolfram-superpowers/, March 2023.

Wu, B., Zhong, Y., Zhang, Z., Huang, G., Liu, X., and Jin, X. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, May 2023.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, Kigali, Rwanda, May 2023.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*, Carlsbad, CA, July 2022.

Zhang, Z., Zhang, A., Li, M., and Smola, A. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations*, Kigali, Rwanda, May 2023.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, New Orleans, Louisiana, December 2023a.

Zheng, L., Yin, L., Xie, Z., Huang, J., Sun, C., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104*, December 2023b.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. Distllm: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24)*, Santa Clara, CA, July 2024.

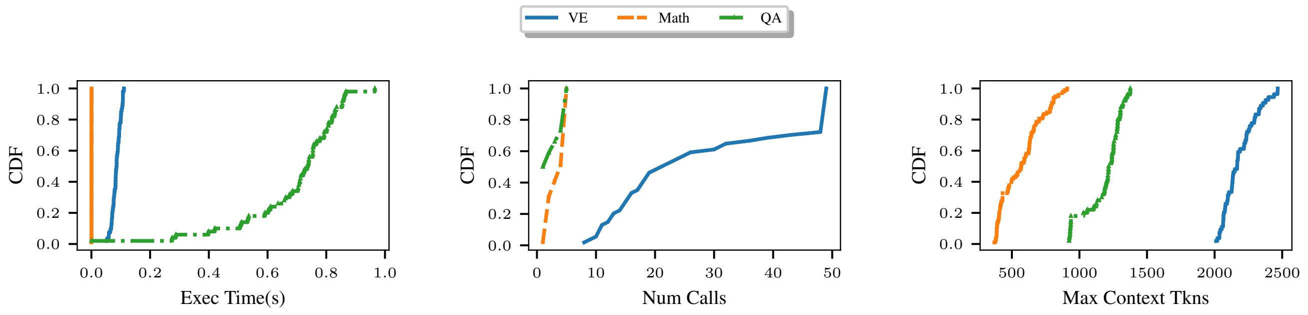

Figure 4: **CDF Results of Short APIs.** *Each line plots the CDF distribution of all the calls of one API type.*

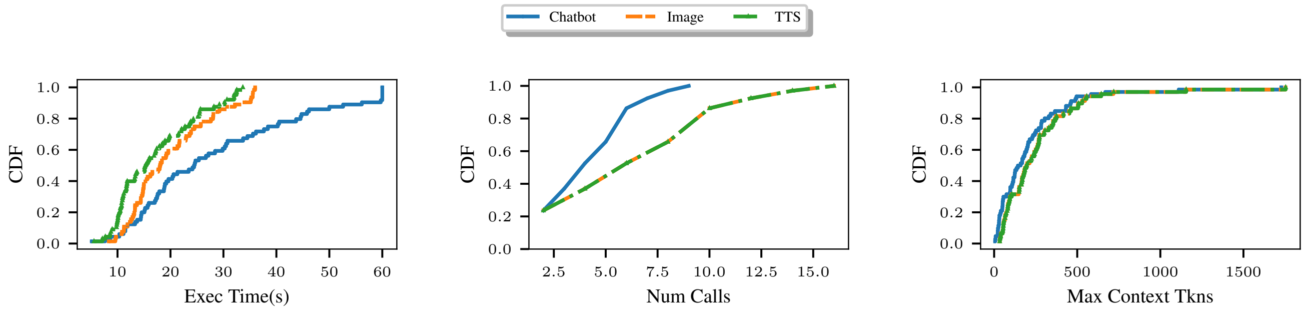

Figure 5: **CDF Results of Long APIs.** *Each line plots the CDF distribution of all the calls of one API type.*

API Properties and Dataset Generation

We analyze the properties of different datasets to evaluate the effectiveness of our inference system. We consider datasets from arithmetic, a knowledge-based question and answering system, a multi-step chat bot, and an embodied virtual environment. The dataset demonstrates different execution time, frequency of API calls, and context lengths. We augment the datasets with relevant APIs from calculator, wikipedia, and a virtual environment gym.

We measure the API execution time, the number of API calls, the number of returned tokens of an API call, and the context length when an API is called. Figures 4 and 5 present the detailed CDF results the four metrics for short-running APIs (math, QA, VE) and long-running APIs (chatbot, image, TTS). Below, we briefly introduce each API type and discuss their properties.

We use the ReAct framework to prompt the LLM to call a Wikipedia endpoint with different questions. The responses from the Wikipedia endpoint are postprocessed. We use the live Wikipedia API endpoint for our benchmark and For each call, we retrieve a summary of the relevant Wikipedia page, which we post-process by limiting the size of the responses to fit within the maximum model sequence length.

To evaluate VE, we prompt GPT-4 LLM with an initial virtual environment and use the ReAct event loop to repeatedly interact with the environment. For our evaluation, we truncate to fit within the context length.

We prompt GPT-4 to generate complex Stable Diffusion prompts at a certain size. Each prompt will trigger an API call of the Stable Diffusion model (Rombach et al., 2021) to generate an image. Each round with the LLM involves a user prompt with normal decoding and then a diffusion call. The number of calls total is modeled around the chat dataset. We estimate the number of image generation rounds (*i.e.*, number of API calls) to be the same as the number of chatting rounds in Chatbot. As there is no standard way of returning tokens from an image generation, we use a short, constant-length sentence describing the generated image as returned tokens. For API execution time, we measure the actual Stable Diffusion call and estimate the human response time in the same as as Chatbot; the sum of them at each round is shown in Table 1 and Figure 5. For the context and return length, we provide a mixture of ShareGPT dataset (Zheng et al., 2023a) and the real prompts.

We use a methodology similar to image generation to understand TTS-augmented LLMs.

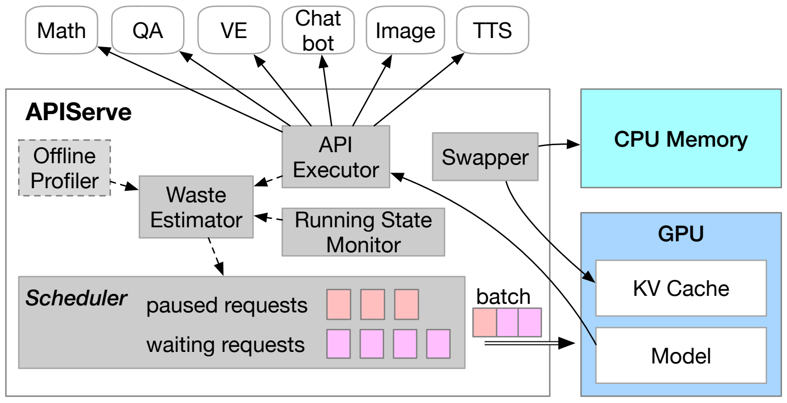

Figure 6: **Overall INFERCEPT Architecture.**

Implementation Details

We implement INFERCEPT on top of vLLM and build INFERCEPT, an inference system designed for API-augmented LLMs. As shown in Figure 6, INFERCEPT includes an API executor that performs different API calls, an iterative-level scheduler that forms a batch of waiting requests and (partial) paused requests, a swap manager that facilitates GPU/CPU memory swapping, a running status monitor that monitors the resource usage and size of running requests, a waste estimator that calculates GPU memory waste for each API-paused request, and an offline profiler that collects basic systems metrics before starting serving.

At the end of each iteration, the scheduler gathers all requests that trigger API calls, adds them to the paused-request queue, swaps as much context (computed keys and values) of paused requests as allowed to CPU memory, determines whether to preserve or discard the remaining context of each paused request based on the GPU memory waste amount calculated by the waste estimator, and calls the actual API. When an API call finishes, INFERCEPT determines how many swapped-out or discarded tokens to swap in or recompute in the next iteration. It also determines how many waiting requests (non-API paused requests) to schedule in the next iteration. When all the context of a paused request has been restored, INFERCEPT adds the tokens returned by the API call to the resumed request.