DroidSpeak: KV Cache Sharing for Efficient Multi-LLM Serving

Yuhan Liu[1]* Yuyang Huang[1] Jiayi Yao[1] Zhuohan Gu[1] Kuntai Du[1] Hanchen Li[1] Yihua Cheng[1] Junchen Jiang[1]
Shan Lu[2] Madan Musuvathi[2] Esha Choukse[2]
[1]University of Chicago [2]Microsoft

Abstract

Large Language Models (LLMs) are increasingly employed in complex workflows, where different LLMs and fine-tuned variants collaboratively address complex tasks. However, these systems face significant inefficiencies due to redundant context processing of the shared context. We propose DroidSpeak, a framework that optimizes context sharing between fine-tuned LLMs derived from the same foundational model. DroidSpeak identifies critical layers in the KV cache and selectively recomputes them, enabling effective reuse of intermediate data while maintaining high accuracy.

Our approach balances computational efficiency and task fidelity, significantly reducing inference latency and throughput bottlenecks. Experiments on diverse datasets and model pairs demonstrate that DroidSpeak achieves up to 3× higher throughputs and 2.6× faster prefill times with negligible accuracy loss compared to full recomputation.

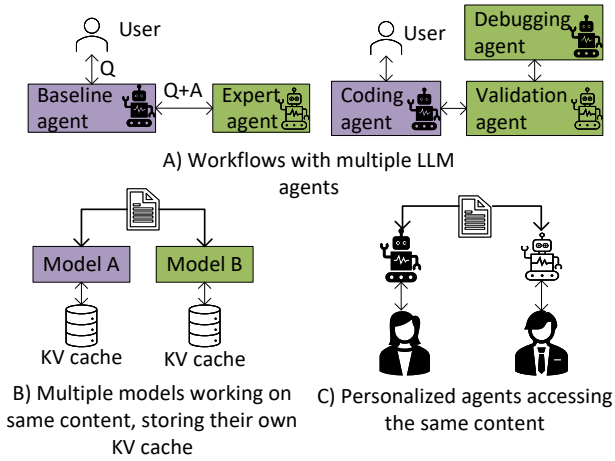

Figure 1: *Various scenarios in which same context is shared by multiple LLMs. DroidSpeak brings down the computation latency by up to 2.6×, increases throughput by 3×, and reduces memory footprint by up to 1.5× in such scenarios.*

1 Introduction

Large Language Models (LLMs) have transformed the landscape of AI-driven applications, enabling a wide range of advanced capabilities, from natural language understanding to complex task automation [62, 84]. The adoption of fine-tuned Large Language Models (LLMs) is accelerating, driven by their ability to tailor foundational models for specific, niche tasks. Fine-tuning enhances LLM performance by adapting them to domain-specific datasets, enabling specialized applications across fields such as healthcare, legal reasoning, customer service, and creative content generation [34,38,83,100]. Several services and prior work offer the ability to serve multiple fine-tuned models at once [15]. At the same time, we are entering a new era where workflows increasingly integrate multiple LLM agents, each fine-tuned for distinct purposes [5, 52, 73]. These agents collectively tackle complex, multi-step tasks, ranging from personalized user experiences to autonomous decision-making [10, 22, 71].

It is increasingly common that fine-tuned LLMs need to share the same contexts in their inputs frequently. For instance, personalized virtual assistants (illustrated in Figure 1) may employ multiple LLMs, each fine-tuned with a user's preference [23, 57], but complement queries with *shared contexts*, such as a common knowledge database. In collaborative robotics, LLM-based agents coordinate task planning and execution by sharing real-time updates [41, 107], as the *shared context* across the LLMs. Similarly, enterprise-level customer support systems might deploy distinct agents fine-tuned for different product categories, requiring them to *share* the context of a customer's chat history, in order to seamlessly ensure consistency in user interactions [6, 95].

The shared context leads to repetitive computations of embeddings and key-value (KV) caches, causing significant inefficiencies. It is well studied that when two LLM inputs share a context as the input prefix, they will each generate the embedding and KV caches of the same context during the prefill phase [40]. Yet, the prefill phase is computationally intensive and accounts for the majority of inference latency, particularly with long contexts [87, 110]. On the other hand, studies have shown that this phase not only consumes substantial power but also reduces system throughput by delaying subsequent queries [50,97,109]. These inefficiencies become a bottleneck when multiple fine-tuned LLMs repeatedly process the same context independently, such as in multi-agent workflows.

Addressing this inefficiency is critical to enable efficient and scalable multi-agent and multi-model systems. In this paper, we focus on the subset of scenarios where fine-tuned LLMs are derived from the same foundational model. This shared origin offers an opportunity to optimize context sharing by reusing intermediate data, such as embeddings and KV caches, between models. However, while this can potentially

[1]Yuhan Liu is affiliated with the University of Chicago, but was a Microsoft intern during this work.

eliminate redundancy, naively reusing the *entire* KV cache causes *substantial* accuracy degradation, as fine-tuning introduces task-specific differences between models. Our core idea is to investigate whether the KV cache generated by one model (the sender) can be shared and *partially reused* by another model (the receiver).

To achieve such a core idea, we leverage insights from prior research indicating that fine-tuning typically updates only a subset of model layers [80,89,103,105,106], leaving many layers structurally similar to the base model. We hypothesize that selectively recomputing critical layers of KV caches while reusing non-critical ones can balance accuracy and efficiency. Through a detailed analysis of KV cache reuse patterns, we confirm this hypothesis and uncover a key challenge: only recomputing the critical layers, whose KV cache has a high impact on LLM outputs, results in both performance inefficiencies and accuracy loss due to compounding errors caused by reused layers between recomputed layers. This insight guides our design of a methodology to identify contiguous chunks of reusable layers, ensuring only one transition from a reuse phase to a recomputation phase.

Building on this foundation, we propose *DroidSpeak*, a novel framework for efficient KV cache reuse across fine-tuned LLMs. DroidSpeak identifies critical layers through offline profiling, enabling sufficient recomputation while reusing as many non-critical layers as possible. By minimizing redundancy and optimizing transition points, DroidSpeak achieves significant improvements in inference latency and system throughput while maintaining high accuracy. We further compare DroidSpeak against alternative approaches, such as deploying smaller models, and demonstrate its advantages in terms of accuracy-efficiency trade-offs. We make the following contributions in this paper:

- We are the first to identify the key inefficiency of repeated prefill-phase computations on shared context across LLMs in modern workflows.
- We perform an in-depth study of KV cache sharing, presenting insights on the patterns.
- We introduce DroidSpeak, a framework optimizing KV cache reuse across LLMs to achieve high accuracy with reduced overhead.
- We evaluate DroidSpeak on diverse datasets and fine-tuned model pairs, demonstrating its ability to reduce latency by up to 2.6×, improve throughput by up to 3×, and in shared KV storage scenarios, reduce memory footprint by up to 1.5×.

Through DroidSpeak, we provide a practical and scalable solution for optimizing multi-agent LLM workflows, paving the way for future innovations in AI systems that combine efficiency with task-specific fidelity.

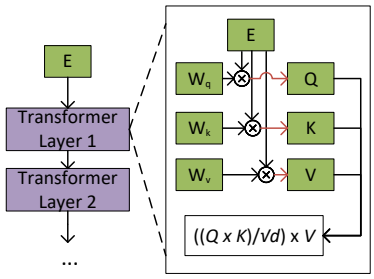

Figure 2: *Illustration of the use of embedding (E), query (Q), key (K), and value (V) tensors in self-attention in transformer-based LLMs.*

2 Background & Motivation

In this section, we give a brief introduction for the background of the emerging workload of context sharing between different fine-tuned model versions and the motivation for DroidSpeak.

2.1 Pertinent Transformer Concepts

The recent wave of generative AI is fueled by the advent of high-performing models that are transformer-based and decoder-only [21, 24, 29]. In this work, we focus on these types of models. **Query, Key, Value, and Embedding:** In transformers, Q (Query), K (Key), and V (Value) are the core components of the attention mechanism [9, 47, 59, 86, 99].

- Query (Q): Represents the vector of the current token to seek relevant information from other tokens in the input sequence.
- Key (K): Encodes attributes of the available data to determine its importance relative to the query.
- Value (V): Contains the actual data or representation that is being passed along.
- Embeddings (E): Is the dense vector representation of input tokens. It maps discrete tokens (e.g., words) into continuous vector spaces, capturing syntactic and semantic relationships between tokens.

An LLM consists of several layers, each with its own Q, K, V, and E. For example, the Llama-3.1-70B model has 80 layers. We denote the K and V vector altogether as KV cache, and the embedding E vector as E cache. Within each layer, embeddings are the starting point for subsequent transformer computations. They will be projected into tensors Q (Query), K (Key), and V (Value) at the beginning of the attention mechanism. Figure 2 illustrates how these components are being used across LLM layers.

The quality of embeddings directly affects the model's ability to understand and process the input context effectively.

Prefill and Decode phases: Large Language Models (LLMs) process input and generate output in two distinct phases: the prefill phase and the decode phase, as shown in Figure 3.

In the ***Prefill Phase***, the LLM processes the entire input

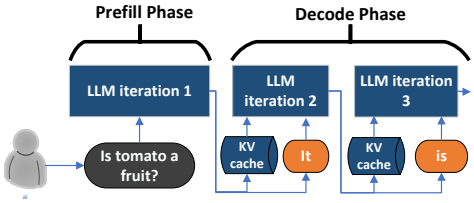

Figure 3: *Prefill and decode phases.*

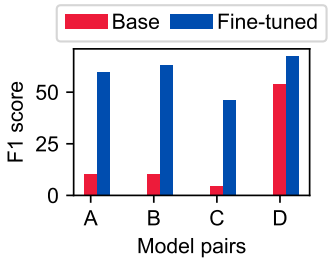

Figure 4: *Fine-tuned model gives higher accuracy than baseline.*

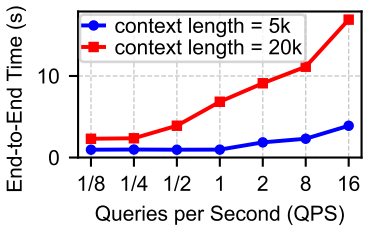

Figure 5: *Shorter input leads to smaller end-to-end time.*

context to compute the embeddings and the KV caches for each token. This phase involves the application of the model's attention mechanism across all layers and the storage of intermediate representations that encode the input context. The prefill phase is computationally intensive, with its complexity scaling quadratically with the input length, making it the dominant contributor to inference latency in long-context scenarios.

In the ***Decode Phase***, the model uses the cached representations (*i.e.,* KV cache) generated in the prefill phase to sequentially produce tokens one by one as the output. This avoids the need to reprocess the entire context. As a result, the decode phase has significantly lower computational overhead compared to the prefill phase, with its complexity scaling linearly with the output sequence length.

Metrics: The main metrics used to capture the performance of LLMs are:

- Time to First Token (TTFT): The duration from query submission to the generation of the first token. This measures the query's queuing delay and the prefill phase duration.
- Time Between Tokens (TBT): The average time between two generated tokens.
- End-to-end latency (E2E): The duration from query submission to the generation of the last token.
- Goodput: The throughput supported by the system while still meeting the SLOs (Service-Level Objectives).

For the accuracy measurement, we used the metrics exposed by various datasets, as described in Sections 3.1.

2.2 Emerging Trends: Context Sharing Across LLMs

Fine-tuned LLMs: Despite being versatile, foundational LLMs can be improved a lot with fine-tuning. Fine-tuning adapts Large Language Models (LLMs) to specific tasks or domains, optimizing their performance for nuanced and specialized queries. For example, a fine-tuned customer support LLM can handle troubleshooting requests with greater accuracy [72], while legal assistant benefits from training on case law and statutes [100]. Recent works like Low-Rank Adaptation (LoRA) [34] have transformed the landscape by introducing a lightweight and efficient approach to fine-tuning. Instead of modifying all model parameters, LoRA applies low-rank updates through parameter adapters, reducing computational and memory overhead.

Figure 4 shows the accuracy comparison of a few pairs of a foundational model and its fine-tuned version on the specific tasks for which the fine-tuning was performed. The four model pairs are Llama-3-70B-Instruct vs Llama-3-70B, Mistrallite vs Mistral-7B, Llama-3-8b-Instruct vs Llama-3-8B, and MAmmoTH2 vs Llama-3-8B, respectively.

Agentic Workflows: Agentic workflows, powered by advances in LLMs, represent a paradigm shift in automation and collaboration [35,44,45,91]. These workflows integrate multiple specialized LLM agents, each fine-tuned for specific tasks, to collaborate and solve complex, multi-step problems. Examples include chatbot ecosystems, where agents handle distinct customer service queries, or collaborative robotics [53,65,90], where LLM agents coordinate tasks like navigation, planning, and execution. This trend stems from the growing need for modular and scalable AI systems that can dynamically adapt to diverse scenarios.

Need for context sharing: In agentic workflows, agents often share a common context, exchanging data and task progress to ensure coherence and consistency. For instance, two agents might work together on summarizing lengthy documents or solving domain-specific queries while building on each other's outputs. Agents also could have a shared conversation history between themselves [67].

Personalized Models: Personalized models tailored to individual users or tasks are increasingly prevalent in AI systems, particularly in applications like chatbots, virtual assistants, and recommendation engines [8, 14, 85].

Need for context sharing: These models often share overlapping contexts, such as common conversation histories or shared knowledge bases, to ensure continuity and relevance. For instance, two assistants answering similar queries about current events may process identical news summaries.

We motivate DroidSpeak with these emerging trends in the workloads today that fuel the need for efficient context sharing across fine-tuned LLMs.

2.3 Prefill interference

Given the distinct features of the prefill and decode phases discussed in Section 2.1, long prefill phases tend to reduce the overall goodput of the inference system. TTFT super-linearly increases with the length of the input. Long prefill phases create delays that ripple into the token generation process due to scheduling challenges. Figure 5 shows the impact of the length of the prefill phase on the E2E latencies of the inference system. We can see that longer inputs can lead to much longer end-to-end latency than shorter inputs.

2.4 Cross-GPU interconnects

With the growing adoption of LLM-driven applications, cloud service providers have significantly expanded their GPU-based offerings, resulting in the deployment of large-scale GPU clusters [4, 19, 54, 61]. These clusters typically consist of machines equipped with 8 flagship NVIDIA GPUs, such as the A100 or H100, and more recently, GB200s.

To ensure efficient communication, GPUs within these clusters are interconnected via high-bandwidth links such as Mellanox InfiniBand networks [56, 60]. This robust data plane network provides bandwidths ranging from 25 to 50 GBps per GPU pair [20, 56] across individual servers, facilitating the rapid transfer of large volumes of data necessary for LLM workloads.

3 Reusing KV cache across the LLMs

Based on the trends described in Section 2.2, we set forth to solve the inefficiency of recomputation around the shared context between fine-tuned models.

In current LLM serving systems, different models need to repeatedly prefill for the same piece of shared context. This inevitably increases the computation and delay during online serving. For example, if three different models need to reuse the same context, the demand for computation will triple and this could also incur a queueing delay when resources are under contention. Even with the more recent KV cache reusing systems like [18, 25], we need to store multiple versions of the KV cache, which uses much more storage space.

One solution is to reuse the KV cache for the shared context across the models. If KV cache can be shared across models, the prefill computation could be reduced for online serving. For the KV cache reuse systems, instead of storing different versions of KV cache for different models [25, 40], we could store the KV cache for one model version and then reuses it across different models.

In this section, we will discuss the properties and patterns we observe around KV cache sharing across different models.

3.1 Building the benchmarks and datasets

Before getting into the KV cache sharing and patterns, we describe the benchmark set we build for DroidSpeak. The study needs pairs of models that share the context provided by the datasets. The following assumptions are also made when building the benchmark.

Specialized Model	Baseline Model
glue_sst2	conllpp
gsm8k	glue_stsb
phi-3.5-mini-instr-adapter	phi-3.5-mini-instr-task15
phi3.5-mini-instr-adapter-v2	phi-3.5-mini-instr-task15
llama-3-8b-sft-lora-ultrachat	fingpt-llama-3-8b
llama-3-8b-chat-lora	fingpt-llama-3-8b
mistrallite	mistral-7b
llama-3.1-70b-instruct	llama-3.1-70b

Table 1: *The model pairs used in our paper. We use three datasets, namely HotpotQA [96], multifieldQA_en [39], and 2wikimQA [30] across all the pairs.*

- The pair of models should share the same foundational model. Specifically, the pair can either consist of the foundational model and a fine-tuned model based on it, or, two fine-tuned models based on the same foundational model.
- The dataset selected should be related to the task for which one of the models has been fine-tuned. This is important since in any context-sharing scenario, the specialized model is performing the specialized task.
- The specialized model fine-tuned on the task in the corresponding dataset should yield better accuracy on the dataset than the other model in the pair.

Using these assumptions, we formulate the benchmark as shown in Table 1. We use 8 pairs of models across 3 datasets (including HotpotQA [96], multifieldQA_en [39], and 2wikimQA [30]). The quality or accuracy metric used is taken directly from the dataset.

Although we discuss several use cases in Section 2.2 and Figure 1, we focus on the use case where the baseline model generates the intermediate state for the context and the specialized model reuses its intermediate states, with both models deployed on separate nodes. This is a challenging use case because *a)* The baseline model has worse accuracy than the specialized model, so achieving high quality must refresh the KV cache properly, and *b)* The communication delay between the separate nodes must be considered.

We will refer to the context-generator model as the ***baseline model***, and the context-user model as the ***specialized model***.

3.2 Empirical insights of KV cache

3.2.1 *Naive reusing is suboptimal*

The first observation is about naively reusing the baseline model's KV cache on the specialized model. Specifically, we observe that:

Insight 1 *Reusing the whole KV cache between models leads to a huge loss in accuracy.*

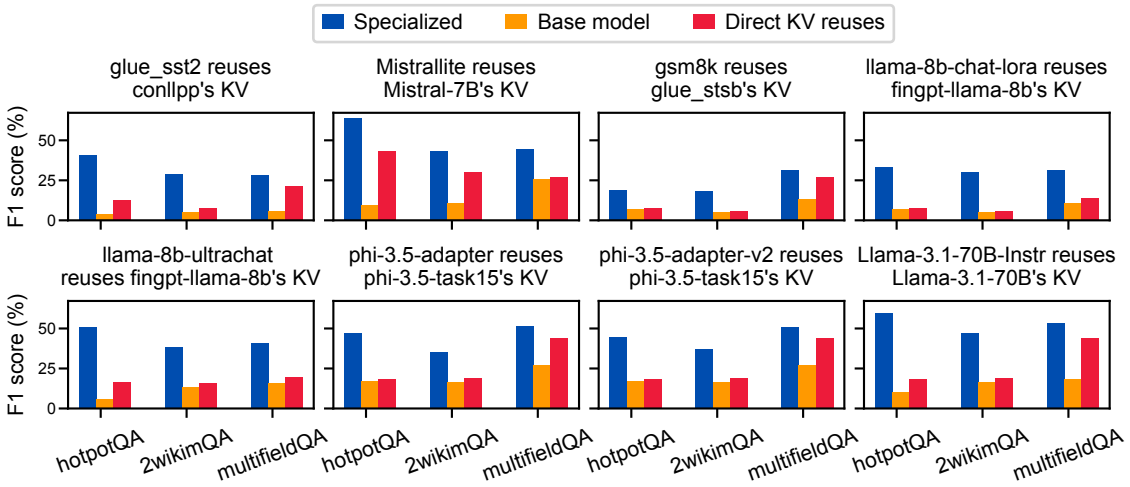

Figure 6: *Directly reusing the full KV cache greatly degrades generation quality.*

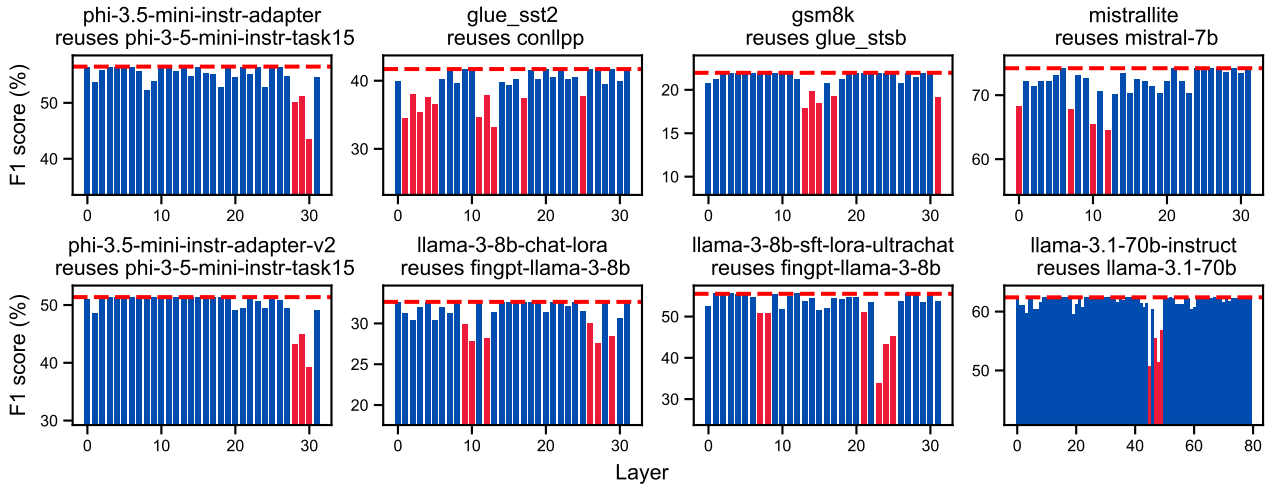

Figure 7: *Different layers have different sensitivities to deviation in KV cache. Plotted by reusing only one layer's KV cache from the base model on the fine-tuned model. The red dashed line is the original accuracy of the fine-tuned model. The bars colored red are those that have an F1 score drop of over 10% compared to the original fine-tuned model.*

A naïve way to reuse the intermediate state between models is to reuse the KV cache as is. In this case, the specialized model receives the KV cache for the whole input prompt from the baseline model. It then uses this to generate the output tokens in the decode phase, thereby completely skipping the prefill phase.

We show the impact of this on accuracy in Figure 6. For each pair of models and dataset, we show the F1 score (higher is better) of *a)* the specialized model, *b)* the specialized model while reusing the KV cache generated by the baseline model, and *c)* the baseline model alone.

Although the accuracy of the specialized model with the baseline model's KV cache is still better than the baseline model alone, we lose a lot of accuracy. HotpotQA tends to lose more than 50% of the accuracy points across all pairs, while the other datasets show varying amounts of changes across model pairs.

3.2.2 *Layer-wise sensitivity to KV cache reuse*

Our second observation is about whether KV cache reusing leads to the same impact across all layers.

Insight 2 *Only a small subset of layers are sensitive to KV cache reusing in terms of accuracy.*

Figure 7 shows the quality drop by reusing part of KVcache from the baseline model. Specifically, each bar represents the quality achieved by the specialized model reusing the KV cache for that corresponding layer from the baseline model, with everything else being recomputed. For most of the model

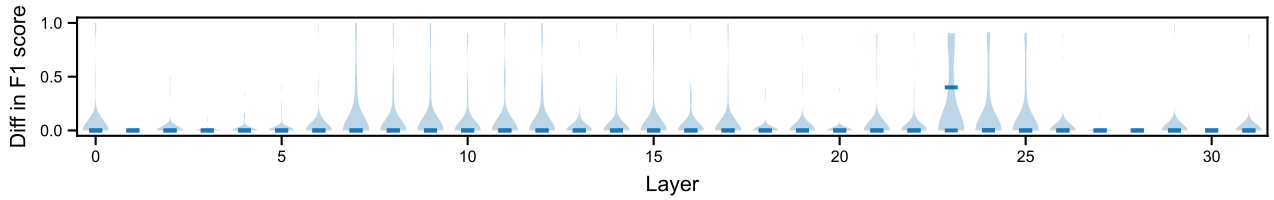

Figure 8: *Variation in F1 score per input within a single dataset (HotpotQA) for model pair Llama-3-8B-sft-lora-ultrachat reusing fingpt-llama-3-8B. We plot the 25 and 75 percentiles. Except layer 23, the 25 and 75 percentiles overlap, indicating a low variance of error sensitivity across all layers except 23.*

pairs, we find only a small subset of layers are sensitive to the deviation in KV cache (*i.e.,* F1 score drops significantly), and we refer to these layers as *critical layers*, and are colored by red. On average across all pairs of models, we identify 11% of layers to be critical. This insight aligns well with the findings from prior works [49, 64, 98], which is *freezing* less critical layers during fine-tuning lead to similar or better quality compared to fine-tune all the layers. Reusing the KV cache from one layer of the baseline model is, at a high level, analogous to freezing the baseline model's weights for that layer. Reusing the KV cache for non-critical layers, similar to freezing these non-critical layers, will have less impact on the output compared to reusing KV for critical layers.

Another important observation is that the critical layers are scattered in different parts of the LLMs. This observation is consistent with prior works in parameter-efficient fine-tuning [103, 105, 106], where they discuss that the layers that contribute the most to the accuracy improvement after fine-tuning may reside in any part of the models.

3.2.3 *Similarity of sensitivity across different inputs*

Our third observation is about whether different inputs show similar patterns in layer-wise sensitivity.

Insight 3 *The variation in KV cache patterns across inputs is only notable for critical layers.*

Figure 8 shows the violin plot of normalized change in F1 score per input in hotpotQA dataset, when llama-3-8b-sft-lora-ultrachat reusing fingpt-llama-3-8b's KV cache of each layer only. Layer 23, which is also marked as the most critical for this model pair in Figure 7 (*i.e.,* the largest F1 score change), shows a wider variation across different data points from the dataset, with a lot of them observing F1 score change $> 50\%$. However, for all the non-critical layers, the variance in the F1 score change is insignificant, meaning that such non-critical layers do not change across various inputs.

This phenomenon is also observed across other model pairs. Intuitively, this can be because critical layers are essential for the reasoning capabilities [17] or the ability to accomplish specific downstream tasks [16]. These reasoning capabilities must remain accurate to interpret any input to the LLMs.

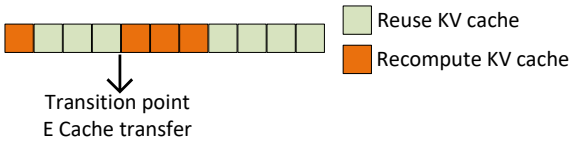

Figure 9: *Illustrating reuse pattern and transition point.*

4 DroidSpeak Design and Implementation

Building on the insights in the previous section, we designed DroidSpeak to enhance the context sharing between two LLMs. The central questions that DroidSpeak targets are the following: *how do we maximize the reuse of KV cache to improve efficiency gains, while keeping the accuracy loss minimal?*

4.1 Challenges with Selective KV Cache Reuse

Insight 2 suggests selectively reusing the KV cache while recomputing it for critical layers *might* preserve accuracy. However, we find that selecting all critical layers scattered across different parts of the LLM is suboptimal for both efficiency and accuracy.

The efficiency challenge: Recomputing critical layers that are non-contiguously placed is inefficient.

During the prefill phase, the output of a layer where the KV cache is reused only contains information about the first generated token. In contrast, recomputing the KV cache needs to start from the E cache of this layer on the whole context. While it is possible to obtain the E cache for the specialized model by performing a full prefill from the context starting from the first layer, this approach completely defeats the purpose of KV cache reuse.

To address this issue, we use the E cache from the baseline model as a proxy to start the recomputing at the layer when transitioning from KV cache reuse to recompute. We refer to this layer as ***transition layer***. As illustrated in Figure 9, for any layer to switch between reuse and recompute, the baseline model must store and transmit the E cache to the specialized model.

The E cache is typically large, reaching up to *twice* the size of the KV cache for the Mistral-7B or Llama-3-8B model families, and up to *four* times larger for the Llama-3.1-70B model family since the KV cache size is optimized by group-query attention [3].

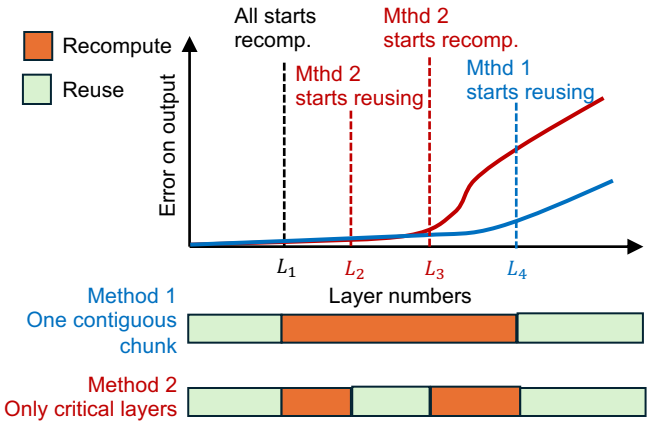

Figure 10: *Illustration of the error brought by each transition point of E cache reusing.*

Consequently, the overhead of storing the E cache in GPU memory and the delays caused by loading it from remote GPU nodes can be substantial, far exceeding the cost of storing and transmitting the KV cache alone.

The accuracy challenge: Furthermore, reusing the baseline model's E cache at the transition layer might also hurt the accuracy of the final output. This is because the E cache loaded from the baseline model (starting point of the recomputation) already differs from the specialized model. Such difference eventually will introduce deviation from the point of recomputation and propagate over all later layers. It is crucial to minimize the error caused by such deviation.

If we pick all the critical layers, which may not appear in contiguous chunks (Figure 7), there will be multiple transition layers from reuse to recompute, introducing multiple deviations in E cache.

Figure 10 illustrates this. If we choose to recompute *only* critical layers (*i.e.,* layers L_1 to L_2, and L_3 to L_4), referred to as method 2 in the figure, we need to load E cache at layer L_1 and L_3. However, whenever we load E cache, the error from E cache will be populated to subsequent critical layers (*e.g.,* loading E cache at layer L_3 populates errors to $L_3 - L_4$) and eventually to the output. Thus, even if all critical layers are recomputed, this will lead to a substantial output error. In contrast, recomputing a contiguous chunk of layers from L_1 to L_4, referred to as method 1 in the figure, avoids this problem by recomputing the KV cache of non-critical layers that are located between critical layers.

To validate this, we compare the accuracy of recomputing a contiguous block of layers versus recomputing the same number of critical layers based on importance, as shown in Figure 11. The latter approach consistently underperforms due to loading multiple E caches from multiple transitions.

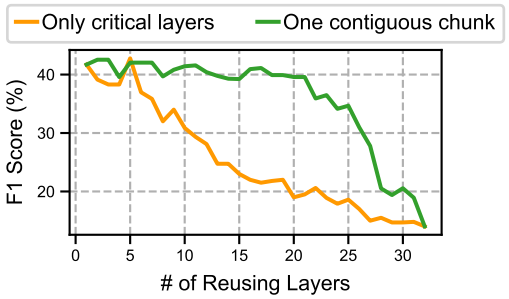

Figure 11: *Accuracy of selective reuse of critical layers vs selective reuse of a contiguous block of layers.*

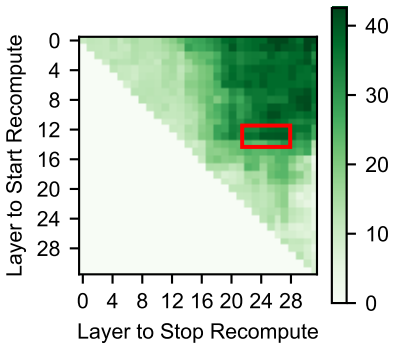

Figure 12: *An example of the accuracy heatmap generated during profiling. For simplicity, we show only reuse→recompute→reuse. Results are shown for glue_sst2 reusing conllpp's KV cache on HotpotQA. The darker the color is, the higher the quality of the generation output is.*

4.2 Reuse patterns considered in DroidSpeak

To address the challenges of selective KV cache reuse, we focus on minimizing the number of loaded E caches from the baseline model. Specifically, we aim for a single transition point of *reuse→recompute*, similar to what we show in method 1 in Figure 10. This design choice balances efficiency and accuracy by limiting the overhead associated with transmitting E cache across multiple transitions and reducing the errors introduced by loading deviated E cache multiple times. We formulate the reuse patterns to be *recompute→reuse→recompute→reuse*, which leads to at most one transition point from reuse to recompute (requiring only one-time loading of E cache). In practice, DroidSpeak also considers two special cases of the aforementioned formulation, *i.e.,* recompute→reuse→recompute, reuse→recompute→reuse.

4.3 Profiling for Optimal Reuse Pattern

A key question still remains: how to determine the optimal transition points from recompute→reuse and from recompute→reuse?

we profile each pair of models to determine the contiguous blocks of critical layers. Our goal is to minimize block size with high accuracy to ensure that we maximize performance

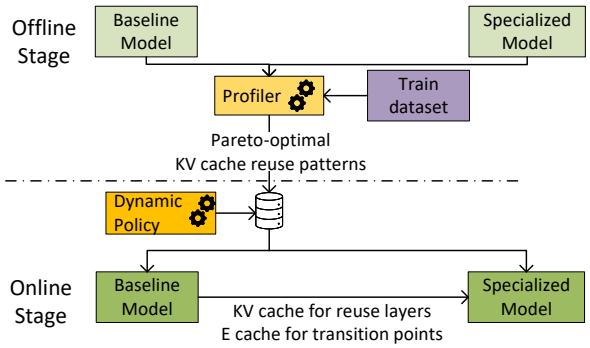

Figure 13: *End-to-end system design.*

gains from reuse.

Figure 12 illustrates an example profiling output for the glue_sst2 and conllpp model pair. Each cell shows the accuracy achieved for different configurations of the reuse→recompute→reuse pattern. The layer number Y on the y-axis indicates the layer at which the switch occurs from reuse to recompute, while the layer number X on the x-axis indicates the layer at which the switch occurs from recompute back to reuse.

Trending toward the diagonal of the plot is optimal for efficiency (i.e., larger Y values and smaller X values), as it minimizes the number of prefill recomputations while maintaining high accuracy. Based on this plot and pattern, the cells within the red-boxed region represent optimal configurations. Among these, we select the leftmost cell in the red region, which involves recomputing layers 13 to 23, achieving near-optimal accuracy while maximizing efficiency.

4.4 DroidSpeak system design

As shown in Figure 13, the design of DroidSpeak consists of two main stages: an offline stage for profiling the reuse pattern and an online stage to execute the partial recomputation with a dynamic workload in mind. During the ***offline stage***, each new pair of baseline and specialized model is profiled as discussed in Section 4.3 with training dataset to find the accuracy-efficiency Pareto-optimal curve of reuse patterns. This step allows DroidSpeak to dynamically choose the right reuse pattern based on available resources. The reuse pattern is stored as a list of three integers to denote recompute→reuse→recompute→reuse.

Dynamic adjustment of the reuse-recompute ratio: By increasing reuse during high-load periods, the system can prioritize throughput by minimizing computational overhead, while during low-load periods for requests with enough slack in latency SLOs, recomputation can be emphasized to maintain higher accuracy.

During the ***online stage***, DroidSpeak dynamically decides which point in the pareto-frontier should be used based on latency SLO. The baseline model stores the E cache for transition points, and transfers KV and E caches as per the current reuse pattern. The specialized model selectively recomputes critical layers while reusing others, achieving a balance between computational efficiency and accuracy.

The separation of profiling and runtime phases allows for adaptability across diverse datasets and task requirements, making DroidSpeak a robust solution for efficient LLM inference. In Section 5.5, we demonstrate that a single profiling run for a pair of models is sufficient to generalize across various datasets.

4.5 Implementation

We implement DroidSpeak with about 2K lines of code in Python, based on PyTorch v2.0, CUDA 12.0, and LMCache 0.1.3 [18]. DroidSpeak operates the LLM inference serving engines through these interfaces:

- `store_kv(KVCache, context, LLM)`, `store_e(ECache, context, LLM)`: We split the KV or E cache into layers, and store it in a key-value store in GPU memory.
- `fetch_kv(context, LLM, layer_id)-> KVCache`, `fetch_e(context, LLM, layer_id)->ECache`: This loads the KV or E cache of the corresponding model for that specific `layer_id`.
- `partial_prefill(recompute_config, context)-> text`: it takes in the recomputation configuration and the context, including which layers to recompute during prefill, and then generates the output text.

We implement these three interface in HuggingFace and LMCache [18]. For `store_kv`, after an LLM generates the KV cache for a piece of context, we calculate the hash of the context text, and put it into the key-value store if the context does not exist in the current store. Before we run the inference for any LLM, we obtain the reuse pattern from the offline profiling (§4.3), which includes the layer numbers for recompute and KV cache reuse. During the online inference stage, we call the `partial_prefill` function, which calls `fetch_kv` for the layers for KV cache reusing, and `fetch_e` at the transition layers. Both `fetch_kv` and `fetch_e` are implemented with `torch.distributed` [74] to fetch KV or E cache from a remote GPU node.

5 Evaluation

The key takeaways from the evaluation are:

- Across three datasets and eight model pairs, DroidSpeak can reduce the prefill latency by 1.7–2.6× without compromising accuracy.
- In the online serving system, DroidSpeak achieves up to 3× improvement in throughput.
- DroidSpeak's profiling of recomputing layers is robust across different datasets and model types.

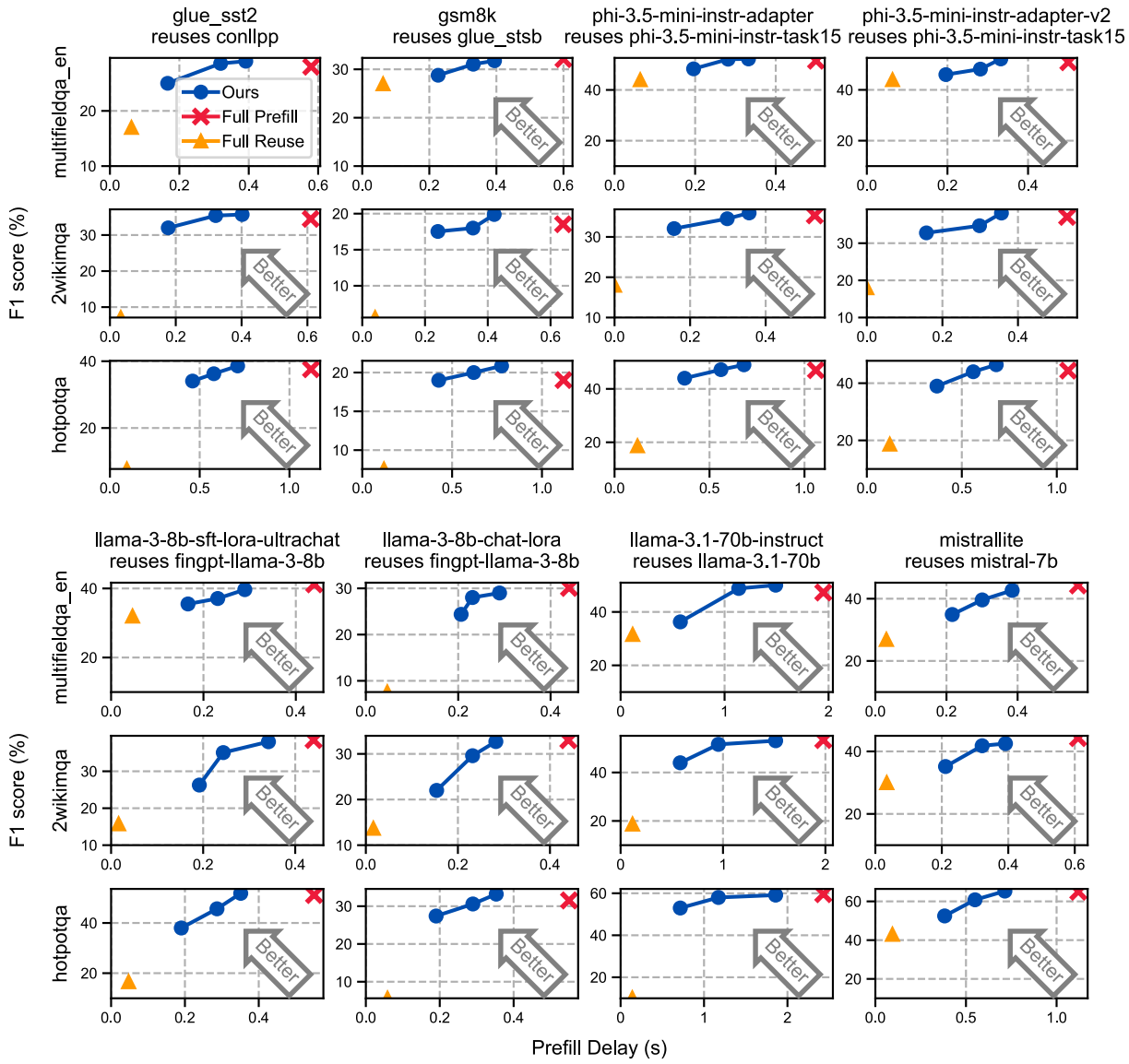

Figure 14: *Prefill delay and F1 score trade-off. DroidSpeak greatly reduces prefill latency while maintaining generation quality.*

5.1 Experiment Setup

Models: We evaluate DroidSpeak on eight pairs of models (Table 1) of different sizes, specifically the fine-tuned versions of Mistral-7B, Llama-3-8B, Phi-3.5-mini and Llama-3.1-70B, selected based on §3.1 These models are fine-tuned on the base foundation model for math reasoning tasks, chat-enhancing tasks, and long context reasoning *et. al.*

Hardware setting: We run the experiments on two A100 virtual machines in Microsoft Azure, namely `Standard_ND96amsr_A100_v4`, which contain 8 80GB A100 GPUs on each virtual machine, and are connected with InfiniBand link.

Datasets: We evaluate DroidSpeak on three different datasets, which consists of 650 contexts in total, and the statistics of the context lengths are shown in Table 2. The tasks are aimed to test LLM's ability in multiple-hop reasoning and multiple-field reasoning from LongBench evaluation suite [7].

Train/test split: As discussed in §4.3, DroidSpeak profiles the minimal block size that maintains accuracy with a "training" dataset offline. Specifically, we use 50 contexts from HotpotQA dataset as the "training" dataset, and use the block size chosen by this training dataset on other datasets in the benchmark. For HotpotQA, we use the other 250 contexts to test in the evaluation section.

Quality metrics: We measure generation quality using the standard metric of each dataset, following prior work [7, 50, 97]. Specifically, we use F1 score, which measures the probability that the generated answer matches the ground-truth

Dataset	Size	Med.	Std.	P95
hotpotQA [96]	300	10933	5160	18650
2wikimQA [7]	200	7466	3976	10705
multifieldQA_en [7]	150	8084	3849	14680

Table 2: *Size and context lengths of datasets in the evaluation.*

answer for the question-answering task.

System metrics: We use the system metrics listed in §2.1 to evaluate DroidSpeak compared with the baselines, including TTFT, TBT, E2E. In §5.2 we also measure prefill latency, which includes the prefill computation time on GPU and the loading delay to fetch KV and E cache through InfiniBand bandwidth link across two GPU nodes.

Baselines: We compare DroidSpeak with the following baselines:

- Full prefill: the receiver model prefills the text of the context with vLLM, which represents the baseline of the highest computation overhead but the best quality we can get.
- Full KV cache reuse: the receiver model directly reuses the KV cache from the sender model, and the receiver model runs decoding with the transferred KV cache.
- Smaller models: In §5.7, we also compare Llama-3.1-70B-Instruct's accuracy and latency trade-off with DroidSpeak with Llama-3.1-8B-Instruct, which is fine-tuned with the same instruct-tuning dataset.

5.2 Lower Latency with Preserved Accuracy

We first demonstrate DroidSpeak's reduction in prefill delay and accuracy trade-off in Figure 14. Across 8 pairs of models on three datasets, DroidSpeak achieves 1.7–2.6× reduction in prefill delay over the full prefill method, without compromising generation quality. On the other hand, when compared with reusing all of sender model's KV cache, DroidSpeak successfully preserves the improved quality of the receiver model despite a slightly higher delay.

Understanding DroidSpeak's improvement: DroidSpeak outperforms the baselines for various reasons. Compared to the full prefill baseline, DroidSpeak achieves significantly lower prefill delay as only a small fraction of layers are prefilled. In contrast to full KV reuse, DroidSpeak has a longer prefill latency because it does not perform prefill at all. However, it greatly reduces accuracy because it misses the opportunity to leverage layer-wise sensitivity in the KV cache.

5.3 Throughput and Latency Improvement

To see the impact of DroidSpeak on improving the throughput of an online LLM inference system, we simulated an online inference scenario by pairing the datasets with request arrival times following uniform distribution under different incoming rates to evaluate the performance of DroidSpeak in more practical workloads.

As demonstrated in Figure 15, we compare the TTFT, TBT, and E2E impact under various request rates on HotpotQA dataset with four pairs of models. For DroidSpeak, we chose the configuration within 1% accuracy drop for these pairs of models.

TTFT: Since the full-recompute baseline has around 2× higher prefill latency than DroidSpeak, the queuing delay affects (knee in the curve) its TTFT at a much lower QPS than what DroidSpeak can support.

TBT and E2E: Although we are only reducing the TTFT directly in DroidSpeak, the second-degree effect through less interference and better scheduling brings down the TBT and E2E latency too, as shown in Figure 15.

Throughput: Assuming an SLO that avoids the effects of high queuing delays on TTFT, TBT, and E2E latency, DroidSpeak can support 2-3× higher throughput as shown in the Figure 15.

5.4 DroidSpeak latency breakdown

DroidSpeak requires moving the context (KV cache and E cache) from the baseline model to the specialized model. To demonstrate the overheads incurred by this transfer with increasing context length, we present Figure 16 with the prefill latency using the model pair glue_sst2 and conllpp on the HotpotQA dataset. We observe that across different input lengths, DroidSpeak consistently reduces the prefill latency by half. This consistent improvement occurs because DroidSpeak skips the prefill computation using the same reuse pattern, resulting in a proportional latency reduction regardless of the input context size.

Furthermore, the loading delay to fetch KV and E cache from another GPU node that is interconnected with InfiniBand link is very low, taking up only at max 11% of DroidSpeak's total prefill delay, which can be overlapped with compute through further optimizations.

5.5 Robustness across datasets

As discussed in §4.4, DroidSpeak profiles the KV cache reuse pattern using a single profiling run on a "training dataset" during the offline stage and then generalizes the profile results to other datasets during the online stage.

Figure 17 illustrates whether the profile obtained on one dataset offline generalizes well to other datasets. In each subfigure, we plot the Pareto frontier of the F1 score versus the number of reused layers, obtained through profiling on the original testing dataset vs two other datasets in our benchmark using glue_sst2 and conllpp model pair.

The figure demonstrates that the Pareto frontier obtained using the profile from the training dataset on the testing dataset closely resembles the frontier obtained using the profile directly from the testing dataset. Across all the pertinent configurations, the maximum difference in the score is 4 points,

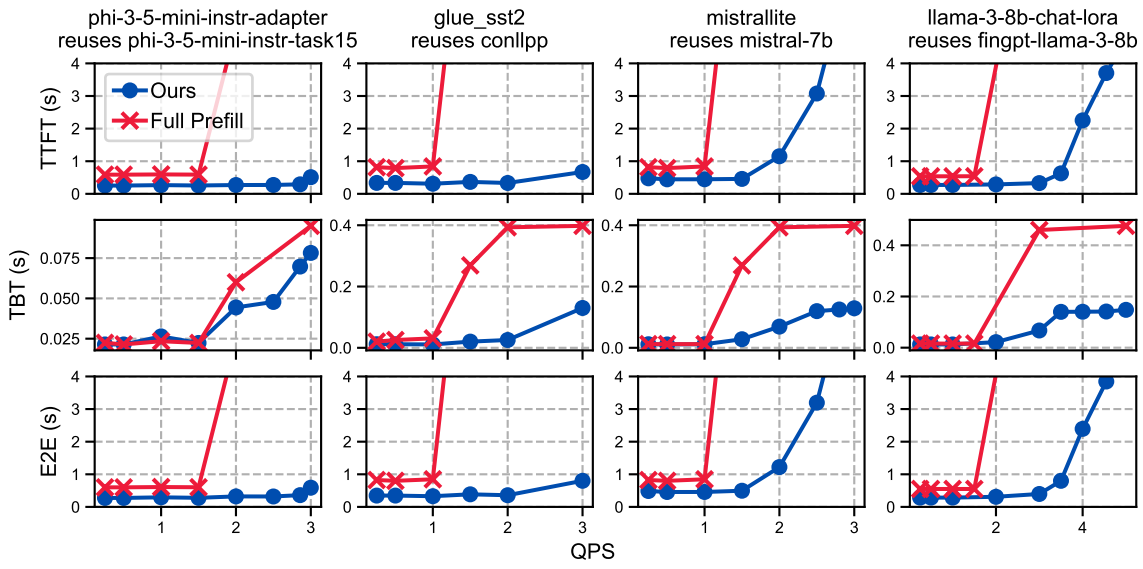

Figure 15: *The impact of arrival rate on Time-to-first-token (TTFT), Time-between-tokens (TBT), and end-to-end latency (E2E), when the DroidSpeak's quality is same as full prefill.*

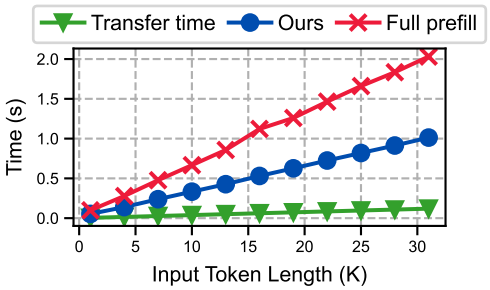

Figure 16: *Impact of context length to DroidSpeak's prefill latency and the loading delay to fetch KV and E cache.*

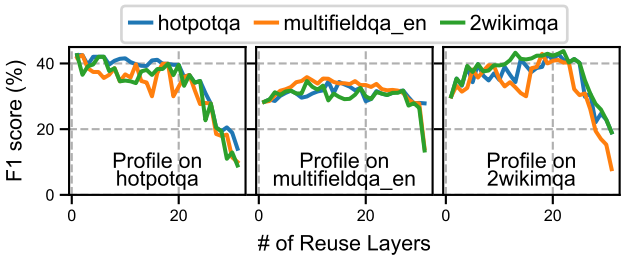

Figure 17: *Using the recompute layers profiled on training datasets works well on testing datasets.*

with the average being 2 points. This result further validates the sufficiency and robustness of our profiling strategy.

5.6 Case study of other model types

So far, we presented results on 3 QA benchmarks from LongBench [7]. To demonstrate that the mechanisms of DroidSpeak apply broadly to other types of models and datasets as well, we apply DroidSpeak on a model pair where the specialized model is fine-tuned on math reasoning, and tested on a task that aims to test LLM's ability in math problem-solving.

In figure 18, we run GSM8K [101] dataset on MAmmoTH2 [101]. Note that the Pareto frontier obtained follows a very similar pattern compared to the LongBench models and dataset, demonstrating the wide applicability of DroidSpeak.

5.7 Comparison against a smaller model

Since DroidSpeak trades off minimal accuracy impact with latency, we compare DroidSpeak on a larger model with a smaller model of the same architecture to show our superior performance in quality and delay trade-off.

In Figure 19, we compare DroidSpeak on Llama-3.1-70B-Instruct and Llama-3.1-8B-Instruct, which is a smaller version of Llama-3.1-70B-Instruct and fine-tuned on the same dataset to enhance the base LLM's ability to follow instructions. As shown, Llama-3.1-8B achieves approximately a 4× reduction in prefill delay but suffers a reduction in F1 score of about half compared to the original F1 score of Llama-3.1-70B-Instruct.

One significant drawback of using a smaller model to achieve speedup is the the overhead of switching between small and large models. For example, when additional resources become available, switching back to the larger model

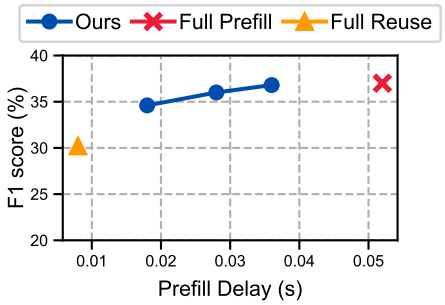

Figure 18: *Prefill delay and accuracy trade-off for MAmmoTH2 (fine-tuned on math reasoning tasks).*

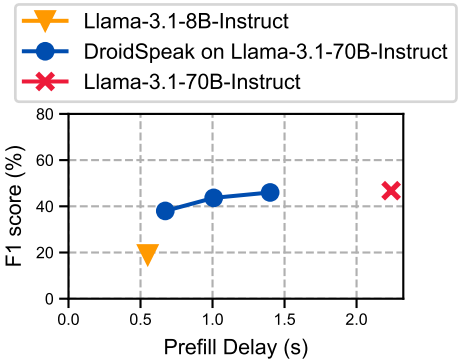

Figure 19: *DroidSpeak applied on Llama-3.1-70B-Instruct has higher accuracy than Llama-3-8B-Instruct.*

to improve serving quality incurs the overhead of loading the larger model back onto the GPU, which can degrade system throughput. In contrast, DroidSpeak can easily adapt to the available compute resources by adjusting the number of layers to be recomputed. This enables more possibilities in efficient scaling up or down on demand.

6 Discussion and Future Work

Recompute for a subset of tokens: In DroidSpeak, we recompute complete chunk of critical layers. The work could be extended, potentially to only recompute for a subset of layers, further reducing latency perhaps.

Reduction in power and energy: Previous work [69, 70, 81] has shown that the prefill phase is much more power-intensive than the decode phase. Decode-heavy tasks in fact, can be run on lower power hardware, since they are memory-bound rather than compute-bound [69]. Since DroidSpeak drastically reduces the prefill phase, it can potentially be used to reduce energy and power of the overall system.

Using KV cache compression: Future work could integrate our approach with KV compression techniques to further reduce memory and transmission costs.

LLMs with different foundational models: Future work could extend our approach to scenarios where models do not share the same foundational model. This would involve developing techniques to align and adapt intermediate representations across structurally different models.

Impact on Security and Responsible AI: Sharing intermediate data across agents can expose sensitive context, increasing the attack surface for adversarial exploitation or unintended data leakage. Moreover, optimizing performance without careful safeguards may inadvertently amplify biases or degrade fairness, raising ethical concerns in AI deployment.

7 Related Work

Fine-tuning: Fine-tuning LLMs for specific tasks has gained importance, but it remains resource-intensive. Methods like LoRA [78, 88] and importance-sampling approaches like LISA [64, 98] address these challenges by enabling parameter-efficient adaptation.

Multi-agent systems: Multi-agent systems show promise in areas such as coding [12, 31, 32, 36, 37, 75, 82], gaming [1, 13, 27, 48, 58, 102, 104], and social simulations [66, 68]. Fine-tuned LLMs as agents improve outcomes in question answering [11], tool learning [77], and personalization [45]. DroidSpeak focuses on reducing communication delays in such systems.

Faster LLM serving: One line of work speeds up LoRA model serving by hosting many LoRA models in memory at the same time. DroidSpeak is faster than them due to the elimination of prefill computation. Other works improve LLM serving including better scheduling [2, 42, 46, 55, 69, 79, 110], memory management for LoRA models [15, 78], and KV cache offloading [25, 33, 40, 43]. All of these works are orthogonal and complementary to DroidSpeak.

Another closely related line of work also trades speed for quality but uses more compact model architectures [51, 76, 92]. However, to smoothly adapt the amount of computation, they need to host multiple models of different sizes in GPU at the same time, which degrades the serving capacity in the system. DroidSpeak does not suffer from it as it simply change the number of recomputed layers.

KV cache optimization: Lots of prior work has focused on optimizing KV caches for a single model. Some work focuses on compressing or offloading KV cache for reduced memory and transmission costs [43, 63, 93, 94, 108]. Another line of research reduces the prefill delay when blending non-prefix KV caches from multiple contexts for the same model [26, 97]. Works such as LLMSteer [28] recalibrate KV caches offline to improve inference quality. Since DroidSpeak focuses on sharing of KV cache across models, these works are orthogonal and can be used in conjunction.

8 Conclusion

In this work, we identified the core challenge of maintaining efficiency in systems where multiple models work on a shared context. We presented DroidSpeak, a framework for optimizing this with KV cache reuse while maintaining high accuracy. Along the way, we identified a critical challenge in managing transition points between reuse and recompute phases, highlighting their impact on performance and accuracy. We demonstrated the robustness of our solution across several model pairs, model types, and datasets. Through selective recomputation and streamlined reuse patterns, DroidSpeak significantly increases throughput (up to 3×), and reduces inference latency (up to 2.6×), while maintaining high accuracy. Our results highlight the potential of DroidSpeak to improve the KV cache efficiency in a shared context multi-model scenario, setting the stage for further innovations in scalable AI systems.

References

[1] Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. Llm-coordination: Evaluating and analyzing multi-agent coordination abilities in large language models, 2024.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024.

[3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[4] Amazon Web Services. Amazon ec2 p5 instances, 2024.

[5] Anonymous. Multiagent finetuning of language models. In *Submitted to The Thirteenth International Conference on Learning Representations*, 2024. under review.

[6] Muhammad Arslan, Saba Munawar, and Christophe Cruz. Sustainable digitalization of business with multi-agent rag and llm. *Procedia Computer Science*, 246:4722–4731, 2024.

[7] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding, 2024.

[8] Mohammad Shafiquzzaman Bhuiyan. The role of ai-enhanced personalization in customer experiences. *Journal of Computer Science and Technology Studies*, 6(1):162–169, 2024.

[9] Gianni Brauwers and Flavius Frasincar. A general survey on attention mechanisms in deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3279–3298, 2021.

[10] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate, 2023.

[11] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023.

[12] Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. Coder: Issue resolving with multi-agent and task graphs, 2024.

[13] Jiaqi Chen, Yuxian Jiang, Jiachen Lu, and Li Zhang. S-agents: Self-organizing agents in open-ended environments, 2024.

[14] Jin Chen, Zheng Liu, Xu Huang, Chenwang Wu, Qi Liu, Gangwei Jiang, Yuanhao Pu, Yuxuan Lei, Xiaolong Chen, Xingmei Wang, et al. When large language models meet personalization: Perspectives of challenges and opportunities. *World Wide Web*, 27(4):42, 2024.

[15] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving, 2023.

[16] Nuo Chen, Ning Wu, Shining Liang, Ming Gong, Linjun Shou, Dongmei Zhang, and Jia Li. Is bigger and deeper always better? probing llama across scales and layers, 2024.

[17] Xinshi Chen, Yufei Zhang, Christoph Reisinger, and Le Song. Understanding deep architecture with reasoning layer. *Advances in Neural Information Processing Systems*, 33:1240–1252, 2020.

[18] Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. Do large language models need a content delivery network? *arXiv preprint arXiv:2409.13761*, 2024.

[19] CoreWeave. Hgx h100 | coreweave, 2024.

[20] CoreWeave Documentation. Hpc interconnect networking | coreweave, 2024.

[21] Chenpeng Du, Yiwei Guo, Hankun Wang, Yifan Yang, Zhikang Niu, Shuai Wang, Hui Zhang, Xie Chen, and Kai Yu. Vall-t: Decoder-only generative transducer for robust and decoding-controllable text-to-speech, 2024.

[22] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023.

[23] Jiabao Fang, Shen Gao, Pengjie Ren, Xiuying Chen, Suzan Verberne, and Zhaochun Ren. A multi-agent conversational recommender system, 2024.

[24] Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R. Costa-jussà. A primer on the inner workings of transformer-based language models, 2024.

[25] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with cachedattention, 2024.

[26] In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference, 2024.

[27] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. Mindagent: Emergent gaming interaction, 2023.

[28] Zhuohan Gu, Jiayi Yao, Kuntai Du, and Junchen Jiang. Llmsteer: Improving long-context llm inference by steering attention on reused contexts, 2024.

[29] Desta Haileselassie Hagos, Rick Battle, and Danda B. Rawat. Recent advances in generative ai and large language models: Current status, challenges, and perspectives, 2024.

[30] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps, 2020.

[31] Samuel Holt, Max Ruiz Luyten, and Mihaela van der Schaar. L2mac: Large language model automatic computer for extensive code generation, 2024.

[32] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.

[33] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. Memserve: Context caching for disaggregated llm serving with elastic memory pool, 2024.

[34] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[35] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2024.

[36] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation, 2024.

[37] Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. Mapcoder: Multi-agent code generation for competitive problem solving. *arXiv preprint arXiv:2405.11403*, 2024.

[38] Cheonsu Jeong. Domain-specialized llm: Financial fine-tuning and utilization method using mistral 7b. *Journal of Intelligence and Information Systems*, 30(1):93–120, March 2024.

[39] Ziyan Jiang, Xueguang Ma, and Wenhu Chen. Longrag: Enhancing retrieval-augmented generation with long-context llms, 2024.

[40] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation, 2024.

[41] Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. Smart-llm: Smart multi-agent robot task planning using large language models. *arXiv preprint arXiv:2309.10062*, 2023.

[42] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[43] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172, Santa Clara, CA, July 2024. USENIX Association.

[44] Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need, 2024.

[45] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal llm agents: Insights and survey about the capability, efficiency and security, 2024.

[46] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 929–945, Santa Clara, CA, July 2024. USENIX Association.

[47] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI open*, 3:111–132, 2022.

[48] Jijia Liu, Chao Yu, Jiaxuan Gao, Yuqing Xie, Qingmin Liao, Yi Wu, and Yu Wang. Llm-powered hierarchical language agent for real-time human-ai coordination, 2024.

[49] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. Autofreeze: Automatically freezing model blocks to accelerate fine-tuning, 2021.

[50] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving, 2024.

[51] Zhenhua Liu, Zhiwei Hao, Kai Han, Yehui Tang, and Yunhe Wang. Ghostnetv3: Exploring the training strategies for compact models, 2024.

[52] Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2410.06101*, 2024.

[53] Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models, 2023.

[54] Meta Engineering Team. Building meta's genai infrastructure, March 2024.

[55] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. Spotserve: Serving generative large language models on preemptible instances, 2023.

[56] Microsoft Documentation. Enable infiniband on azure virtual machines, 2024.

[57] Paul Mineiro. Online joint fine-tuning of multi-agent flows, 2024.

[58] Manuel Mosquera, Juan Sebastian Pinzon, Manuel Rios, Yesid Fonseca, Luis Felipe Giraldo, Nicanor Quijano, and Ruben Manrique. Can llm-augmented autonomous agents cooperate?, an evaluation of their cooperative capabilities through melting pot, 2024.

[59] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[60] NVIDIA Corporation. Nvidia infiniband solutions, 2024.

[61] NVIDIA Newsroom. Nvidia spectrum x: The ethernet networking colossus for ai and accelerated computing, 2024.

[62] OpenAI. Gpt-4 technical report, 2024.

[63] Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns, 2024.

[64] Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning, 2024.

[65] Keivalya Pandya and Mehfuza Holia. Automating customer service using langchain: Building custom open-source gpt chatbot for organizations, 2023.

[66] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery.

[67] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

[68] Joon Sung Park, Lindsay Popowski, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Social simulacra: Creating populated prototypes for social computing systems. 2022.

[69] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting, 2024.

[70] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. Polca: Power oversubscription in llm cloud providers, 2023.

[71] Chau Pham, Boyi Liu, Yingxiang Yang, Zhengyu Chen, Tianyi Liu, Jianbo Yuan, Bryan A. Plummer, Zhaoran Wang, and Hongxia Yang. Let models speak ciphers: Multiagent debate through embeddings, 2024.

[72] Predibase. Predibase finetuning for customer service. https://predibase.com/customer-service-automation, Dec 2023. Accessed: 2024-12-09.

[73] Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents, 2024.

[74] PyTorch Contributors. *Distributed Communication Package - torch.distributed*, 2024. Accessed: 2024-12-10.

[75] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2024.

[76] Anthony Sarah, Sharath Nittur Sridhar, Maciej Szankin, and Sairam Sundaresan. Llama-nas: Efficient neural architecture search for large language models, 2024.

[77] Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. Small llms are weak tool learners: A multi-llm agent, 2024.

[78] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-lora: Serving thousands of concurrent lora adapters, 2024.

[79] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. Fairness in serving large language models, 2024.

[80] Krishna Prasad Varadarajan Srinivasan, Prasanth Gumpena, Madhusudhana Yattapu, and Vishal H Brahmbhatt. Comparative analysis of different efficient fine tuning methods of large language models (llms) in low-resource setting. *arXiv preprint arXiv:2405.13181*, 2024.

[81] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. Dynamollm: Designing llm inference clusters for performance and energy efficiency, 2024.

[82] Microsoft AutoGen Team. Autogen 0.2 documentation - agentchat auto feedback from code execution. https://microsoft.github.io/autogen/0.2/docs/notebooks/agentchat_auto_feedback_from_code_execution, 2024. Accessed: 2024-10-14.

[83] Robert Tinn, Hao Cheng, Yu Gu, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Fine-tuning large neural language models for biomedical natural language processing. *Patterns*, 4(4):100729, 2023.

[84] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[85] Athanasios Valavanidis. Artificial intelligence (ai) applications. *Department of Chemistry, National and Kapodistrian University of Athens, University Campus Zografou*, 15784, 2023.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[87] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 640–654, 2024.

[88] Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 911–927, Santa Clara, CA, July 2024. USENIX Association.

[89] Feijie Wu, Zitao Li, Yaliang Li, Bolin Ding, and Jing Gao. Fedbiot: Llm local fine-tuning in federated learning without full model. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3345–3355, 2024.

[90] Pengying Wu, Yao Mu, Kangjie Zhou, Ji Ma, Junting Chen, and Chang Liu. Camon: Cooperative agents for multi-object navigation with llm-based conversations, 2024.

[91] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023.

[92] Wenhan Xia, Hongxu Yin, and Niraj K. Jha. Efficient synthesis of compact deep neural networks, 2020.

[93] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads, 2024.

[94] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024.

[95] Yingxuan Yang, Qiuying Peng, Jun Wang, and Weinan Zhang. Multi-llm-agent systems: Techniques and business perspectives. *arXiv preprint arXiv:2411.14033*, 2024.

[96] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018.

[97] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion, 2024.

[98] Kai Yao, Penglei Gao, Lichun Li, Yuan Zhao, Xiaofeng Wang, Wei Wang, and Jianke Zhu. Layer-wise importance matters: Less memory for better performance in parameter-efficient fine-tuning of large language models, 2024.

[99] Catherine Yeh, Yida Chen, Aoyu Wu, Cynthia Chen, Fernanda Viégas, and Martin Wattenberg. Attentionviz: A global view of transformer attention, 2023.

[100] Shengbin Yue, Wei Chen, Siyuan Wang, Bingxuan Li, Chenchen Shen, Shujun Liu, Yuxuan Zhou, Yao Xiao, Song Yun, Xuanjing Huang, and Zhongyu Wei. Disclawllm: Fine-tuning large language models for intelligent legal services, 2023.

[101] Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. Mammoth2: Scaling instructions from the web, 2024.

[102] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. Proagent: Building proactive cooperative agents with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17591–17599, Mar. 2024.

[103] Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning, 2023.

[104] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models, 2024.

[105] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. Loraprune: Structured pruning meets low-rank parameter-efficient fine-tuning, 2024.

[106] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023.

[107] Yang Zhang, Shixin Yang, Chenjia Bai, Fei Wu, Xiu Li, Zhen Wang, and Xuelong Li. Towards efficient llm grounding for embodied multi-agent collaboration. *arXiv preprint arXiv:2405.14314*, 2024.

[108] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023.

[109] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024.

[110] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving, 2024.