DroidSpeak: KV Cache Sharing Across Fine-tuned Model Variants

Yuhan Liu[1] Yuyang Huang[1] Jiayi Yao[1] Shaoting Feng[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

Compound AI systems, such as agentic systems, are an emerging trend in large-scale enterprise settings, with multiple LLMs specialized for different users, tasks, and/or roles working together. In these scenarios, different models often process inputs that share the same context prefix. Although much work was done in the past to enable the reuse of prefix KV caches across inputs for a single model, how to enable one model to reuse the prefix KV caches of a different model remains an open question.

We introduce DroidSpeak, the first distributed LLM inference system that enables KV cache reuse across distributed nodes running inference of different LLMs, so long as the LLMs have the same architecture. We present the first study that aims at understanding the impact of sharing KV caches across different LLMs, and if/when such sharing affects quality. Inspired by the findings, we present DroidSpeak, which selectively recomputes a few layers of the KV cache produced by another LLM and reuses the remaining layers, with negligible quality loss. Moreover, carefully pipelining the layer-wise re-computation and the loading of reused KV cache further improves the inference performance. Experiments on diverse datasets and model pairs demonstrate that DroidSpeak achieves up to 4× throughput improvement and about 3.1× faster prefill (time to first token), with negligible loss of quality in F1 scores, Rouge-L or code similarity score, compared to the baseline which does not allow any sharing across models.

1 Introduction

Nowadays, LLM inference has become one of the most resource-consuming workloads in industry, demanding ever larger clusters of GPU machines [5, 75, 90–92, 111]. To reduce the computation demand, a common optimization is for GPU machines that run the *same* LLM to share KV caches of reused input prefixes over the network [22, 45, 57, 65]. However, how to make that optimization work across *different* LLMs is yet to be studied.

Indeed, an emerging trend is hosting multiple *different* LLMs in one GPU cluster. The reason is that with LLMs used in more complex or personalized tasks, multiple LLMs, often fine-tuned from the same foundational model, are needed to serve different users, tasks, or roles. These LLMs work together to perform one complex task or to offer different customized services [10, 14, 31, 69, 98]. Since standard prefix-caching techniques work only when the KV cache is reused by the same LLM, the use of multiple LLMs poses a direct challenge — how to reuse KV cache efficiently across different LLMs in distributed settings.

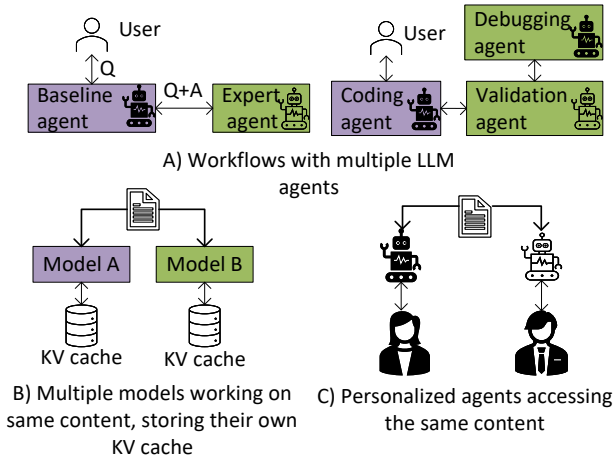

Figure 1: *Various scenarios in which same context is shared by multiple LLMs. DroidSpeak brings down the computation latency by up to 3.1×, increases throughput by 4×.*

To motivate this challenge, we highlight three common use cases of multiple fine-tuned models working together in a system. The first one is *multi-agent systems*, which use different fine-tuned models to serve as the agents and to accomplish collaborative tasks [16, 43]. The second use case is serving *multi-LoRA or multiple fine-tuned models*, such as models that are continuously updated over time with newer data [50], or concurrently serving LoRA adapters [20, 83, 97]. The third use case is *personalized assistant systems*, where a model is fine-tuned for each user (or user type) according to their personal preferences in coding or writing.

In all of these scenarios, prefix sharing is common. For example, in multi-agent systems (Figure 1A), when the coding agent talks to the validation agent, the conversation history of the coding agent will be prepended to the input of the validation agent, to ensure the coherence of the conversation. In a multi-LLM or multi-LoRA inference system (Figure 1B), an updated model in a chatbot application could refer to the same piece of conversation history produced by an older version of the model. In a personalized assistant system (Figure 1C), where each assistant is fine-tuned for a different user's preference, the same news (*i.e.,* the query prefix) can be used to answer different users' queries.

Based on the observations above, we pose the question: *can we share the intermediate states (i.e., KV cache) produced*

by one LLM on a given context to accelerate the prefill for another LLM? In this paper, we seek to answer this question under the assumption that the two LLMs have different weights but the same architecture.

Intuitively, the potential performance benefit of such intermediate state sharing is huge — previous work on single-LLM KV cache sharing has shown up to 8× latency and throughput improvement by speeding up the expensive prefill phase of LLM inference, particularly for long context workloads [34, 35, 52, 65]. However, just like a video decoder could not decode a video encoded with another codec, naively sharing the KV cache across different LLMs would cause generation quality to drop greatly.

Our hypothesis is that there should be a way to *re-compute a small portion and reuse a large portion*, although not all, of the KV cache between two models that are *fine-tuned from the same base model* — since the models are only *fine*-tuned, they should share similar understanding of the same input context and hence help accelerate each other's inference. Of course, the challenge is to validate this hypothesis, and to figure out which part to re-compute or re-use without incurring much delay overhead or degradation of quality.

Through a thorough empirical study (§3.2), we measured eight representative model pairs and found that only a small subset, often around 10%, of layers are sensitive to the KV cache difference between two models in a pair. The identities of these layers vary with different model pairs, but are largely consistent across different inputs for the same model pair. We refer to these layers as *critical layers* in this paper. Therefore, for each pair of LLMs, we propose to selectively recompute these critical layers in the KV cache.

We build our insights into *DroidSpeak*, the first distributed multiple-LLM inference system that enables efficient sharing of KV caches across different LLMs. First, DroidSpeak identifies the critical layer groups through offline profiling on a held-out training set, ensuring sufficient re-computation for accuracy purposes while reusing as many layers' KV cache as possible. Second, DroidSpeak implements smart KV cache loading, which pipelines the loading of KV cache with the re-computation of critical layers, to hide the loading delay from remote nodes as much as possible.

We should note that the high-level idea of selectively re-computing KV cache is not exactly new. DroidSpeak differs with prior work [34, 35, 65, 100] in *why* and *how* to re-compute KV cache: DroidSpeak is designed for KV cache reuse across *different* LLMs, while prior work assumes a *single* LLM; due to the different purposes, none of the prior work chooses to re-compute or re-use a group of layers like in DroidSpeak. For example, CacheBlend [100] updates a reused KV cache, but it still feeds the KV cache to the *same* LLM that generated the KV cache. Moreover, it updates the KV cache of certain tokens, rather than certain layers like in DroidSpeak.

We evaluate DroidSpeak on six datasets across *three* different tasks, including *question answering, text summarization, and coding*, across eight different model pairs. We compare DroidSpeak with various baselines, including direct KV reuse [34], CacheBlend [100], and smaller models. Across these setups, we can reduce the latency by up to 3.1×, improve throughput by up to 4× with negligible drop in quality (measured in F1 score, Rouge-L, or code similarity score).

While the concept of "translating" KV caches between models involves a machine-learning challenge, our primary contribution lies in making it practical in a *distributed system* setting, which requires the transfer and computation of such KV-cache translation to be done efficiently. Specifically, we identify a key system-level insight: once the E cache of the transition layer has been transferred, selective re-computation can start independently of the KV cache transfer for reused layers. This decoupling allows transfer and re-computation to be pipelined efficiently.

2 Background & Motivation

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

2.1 Basic Transformer Concepts

The recent wave of generative AI is fueled by the advent of high-performing models that are transformer-based and decoder-only [28, 33, 40].

Query, Key, Value, and Embedding: In transformers, Q (Query), K (Key), and V (Value) are the core components of the attention mechanism [15, 62, 71, 89, 103]. An LLM model comprises many layers. Each layer generates its E/Q/K/V tensors given an input (Figure 17). We denote the K and V tensors altogether as ***KV cache***, and the embedding E tensors as ***E cache***. Within each layer, embeddings E are the starting point for subsequent transformer computations (including attention).

The quality of E/Q/K/V tensors directly affects the model's ability to understand and process the input context effectively.

Prefill and Decode phases: LLMs process input and generate output in two distinct phases: the prefill phase and the decode phase. In the ***Prefill Phase***, the LLM processes the entire input context to produce the embeddings and the KV caches across all layers. In the ***Decode Phase***, the model uses the KV cache generated in the prefill phase to autoregressively produce tokens one by one as the output.

Fine-tuned LLMs: Despite being versatile, foundational LLMs' capabilities on specific tasks can improve through fine-tuning on specialized domain data. For example, one can turn a foundational LLM into a customer-support agent by finetuning on troubleshooting requests [76], or into a legal assistant by finetuning on case law and statutes [106]. Fine-tuning can greatly improve the accuracy of an LLM on a target domain, as shown in Figure 2(a),

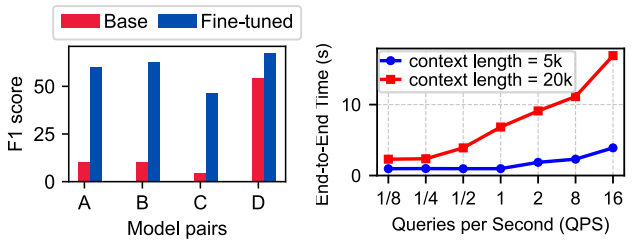

Figure 2: *(a) Fine-tuned model produces higher quality than baseline. (b) Shorter input leads to smaller end-to-end time.*

where the fine-tuned models (`Llama-3-70B-Instruct` [6], `Mistrallite` [104], `Llama-3-8B-Instruct` [6], and `MAmmoTH2` [107]) greatly outperform the foundational models they originate from, on the HotpotQA dataset [99].

Recent works in parameter efficient fine-tuning, like LoRA [46] have made fine-tuned models even more accessible by updating part of the model weights, reducing computational and memory resources during fine-tuning.

2.2 Context Sharing Across LLMs

In compound AI systems, prefix contexts[1] are often shared across different LLMs—either to enhance the coherence of the chat experience or to reference the same set of background documents. In this paper, for convenience, we use the following terminology:

- ***Sender*** model produces the KV cache of a context;
- ***Receiver*** model reuses the KV cache (with limited recomputation) of the reused context.

We describe several concrete use cases of context sharing between different LLMs below.

Agentic Workflows: Agentic workflows represent a paradigm shift in automation and collaboration in the LLM space [4,8,36,48,54,59,95]. These workflows integrate multiple specialized LLM agents, each fine-tuned for specific tasks, to collaborate and solve complex, multi-step problems, or let them play different roles in agent debating. For example, many prior works show that a single LLM is not enough to accomplish complex tasks since it lacks diversity in learned data distribution or the ability to accomplish different specialized tasks, thus multiple fine-tuned LLMs, are needed to collaborate together to solve them [32,58,86,102]; and in CAMEL, a very popular multi-agent framework, different agents can be models fine-tuned on independent data domains [47].

Compared with using different prompts on the same model for different agents, using fine-tuned models as different agents can better improve output quality [16,85].

Need for context sharing: In agentic workflows, different agents often share a common context, often the conversation history of other agents, to ensure the coherence and consistency between agents [39,43,94]. As a concrete example, in coding agentic workflows with a coding and a testing agent, the testing agent (receiver model) has to read both the input instructions and generated code from the coding agent (sender model) to write appropriate unit tests to meet user's needs [43,79,94]. We evaluate context sharing using a realistic trace from a LangGraph-based multi-agent application [3]. In this workflow, four agents collaborate to generate research ideas from English prompts. We use the inputs from Deep Research Bench [29]. On average, 49% of tokens in each conversation are *prefix reuses*, showing that context sharing is common in *real-world* multi-agent workflows.

Personalized Models: Personalized models tailored to individual users or tasks are increasingly prevalent in LLM systems, particularly in applications like chatbots, virtual assistants, and recommendation engines [13,19,88]. In these applications, different assistants are typically LLMs fine-tuned for different users' preferences [44,59]. For example, the personal assistant for a software engineer can be fine-tuned to generate high-quality and concise code snippets, while the personal assistant for a financial analyst identifies marketing angles in the documents without technical detail.

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 will process the same top news.

Multiple-LLM or multi-LoRA serving: In chatbot applications, LLMs often require continuous updates to incorporate new information to provide up-to-date support and higher-quality answers for users [9,26]. As an example, ChatGPT APIs release new API versions based on the same foundational model about every two months, which fine-tunes on the emerging new data [68]. Furthermore, multiple LoRA adapters often need to be concurrently served [20,83] to accomplish different tasks or serve different users.

Need for context sharing: In this case, the updated model (receiver model) often needs to re-process the same sets of popular contexts processed by the older model (sender model) before. Multiple LoRA adapters can share their KV cache when processing the same context.

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 Distributed LLM Inference Systems

As the demand for LLM inference continues to grow, it has become common to serve LLMs in a cluster of GPU nodes. Many companies have developed their own distributed inference systems, such as vLLM Production Stack [90], NVIDIA Dynamo [5], and ByteDance AIBrix [91]. Among all these frameworks, KV cache sharing across nodes is one of the most important features for reducing prefill computation and

[1]Since a shared context is often the prefix of different inputs, we use *prefix* (caching/sharing) and *context* (caching/sharing) interchangeably.

increasing overall throughput. Specifically, when there are multiple requests to the *same model* querying a common prefix context, these systems can transfer KV cache generated by another user request, either from another GPU node or through a centralized storage backend [22, 34, 53].

However, current distributed inference systems have not optimized for multiple-LLM inference yet–they do not explore the opportunity to share KV cache across GPU nodes when the requests are querying *different models*.

2.4 Prefill interference

Given the distinct characteristics of the prefill and decode phases described in Section 2.1, lengthy prefill phases can significantly reduce an inference system's goodput—defined as the number of queries processed per second within a latency SLO [111]. This reduction occurs because TTFT (time to first token) grows super-linearly with input length, and the decoding phase cannot start until the prefill phase completes. Consequently, long inputs often turn prefill delays into the end-to-end bottleneck. For example, as shown in Figure 2(b)—where Llama-3-8B runs on a single A100 GPU with synthetic input lengths of 5K and 20K tokens—setting a 3-second latency SLO reveals that increasing the input size by only 4× can reduce goodput by as much as 32×.

3 Reusing KV cache across LLMs

A simple way to eliminate the overhead of repetitively re-computing the KV cache that another LLM has generated before is to reuse the KV cache produced by another model. As a concrete example, on an A100 GPU and using `Llama-3.1-8B-Instruct`, reusing the KV cache for a 40K-token input can reduce prefill latency from 4s to 0.08 seconds. This naturally leads to the key question: What effect does directly reusing another LLM's KV cache have on generation quality?

In this section, we present the first empirical study of how reusing another model's KV cache impacts output quality, and we further examine whether these effects vary across individual layers of the cache.

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.

- 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 receiver model is performing the specialized task.
- The receiver model fine-tuned on the task in the corresponding dataset should yield better quality than the sender model in the pair.

Using these assumptions, we formulate the benchmark as shown in Appendix Table 1. We use 8 pairs of models across 6 datasets (including HotpotQA [99], multifieldQA_en [110], 2wikimQA [41], multi_news [30], lcc [38], and repobench-p [64]). The receiver models are fine-tuned on datasets spanning chat, coding, instruction tuning, and long-context tasks. The quality metric used is taken directly from the dataset.

We focus on the use case where the sender model generates the intermediate states for the context and the receiver model reuses its intermediate states. This is a challenging use case because the sender model has worse accuracy than the receiver model, so achieving high quality requires properly refreshing the KV cache.

3.2 Empirical insights of KV cache

Naive reusing is suboptimal: The first observation is about naively reusing the sender model's KV cache on the receiver model. Specifically, we observe that:

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

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

We show the impact of this on quality in Figure 3. For each pair of models and dataset, we show the F1 score (higher is better) of *a)* the receiver model, *b)* the receiver model while reusing the KV cache generated by the sender model, and *c)* the sender model alone.

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

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 reuse in terms of accuracy.*

Figure 4 shows the quality drop by reusing part of KV cache from the sender model. Specifically, each bar represents the quality achieved by the receiver model reusing the KV cache for that corresponding layer from the sender model, with everything else being recomputed.

For most of the model pairs, we find only a small subset of layers are sensitive to the deviation in KV cache (*i.e.,* F1 score

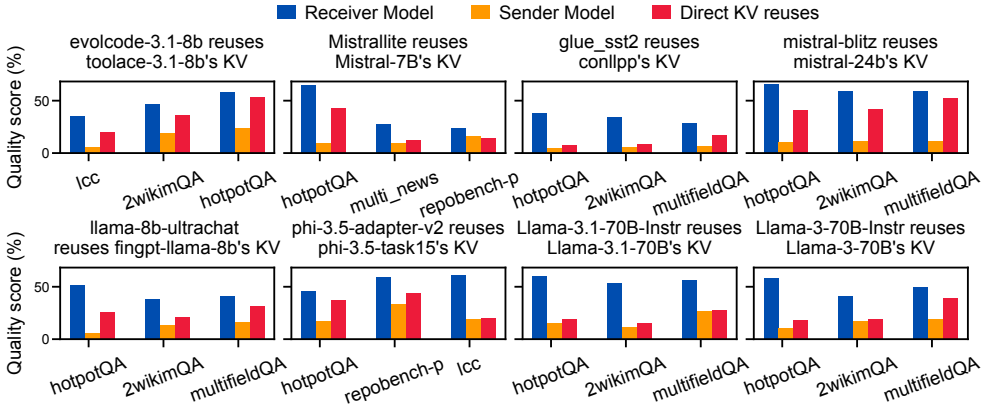

Figure 3: *Directly reusing the full KV cache greatly degrades generation quality.*

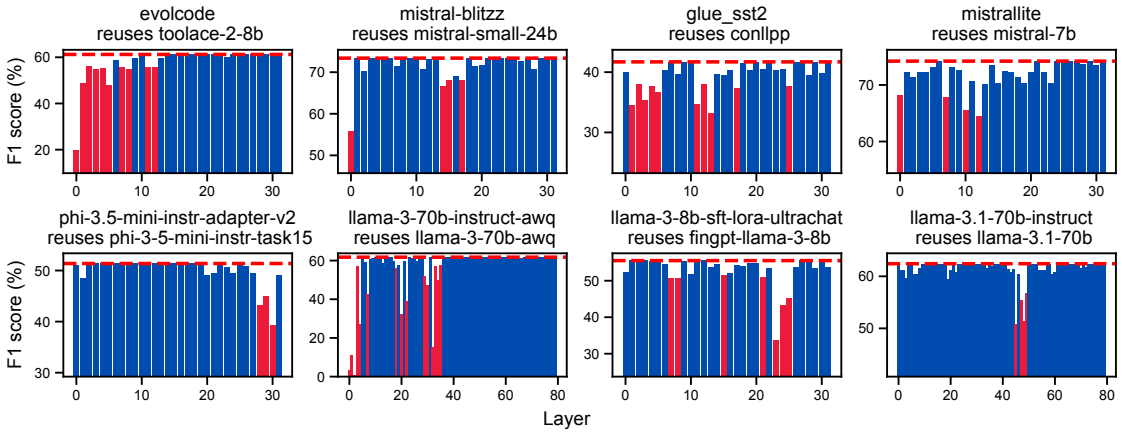

Figure 4: *Different layers have different sensitivities to deviation in KV cache. Plotted by reusing only one layer's KV cache from the sender model on the receiver model. The red dashed line is the original accuracy of the receiver model. The bars colored red are those that have an F1 score drop of over 10% compared to the original receiver model.*

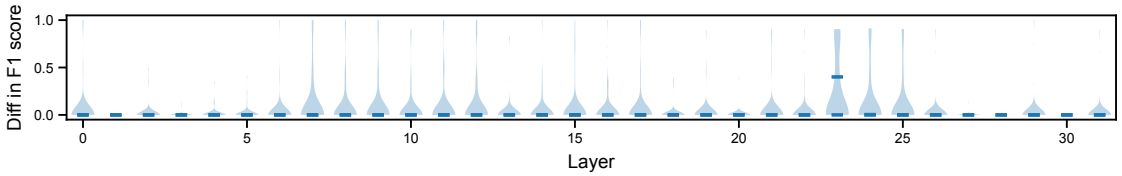

Figure 5: *Variation in F1 score per input within a single dataset (HotpotQA) for model pair Llama-3-8B-sft-lora-ultrachat reusing fingpt-llama-3-8B. We plot the 25 and 75 percentiles. Except layer 23, the 25 and 75 percentiles overlap, indicating a low variance of error sensitivity across all layers except 23.*

drops significantly), and we refer to these layers as *critical layers*, which are colored red. On average across all pairs of models, we identify 11% of layers to be critical.

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 5 shows the violin plot of the 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 4 (*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 [24] or the ability to accomplish specific downstream tasks [21]. These reasoning capabilities must remain accurate to interpret any input to the LLMs.

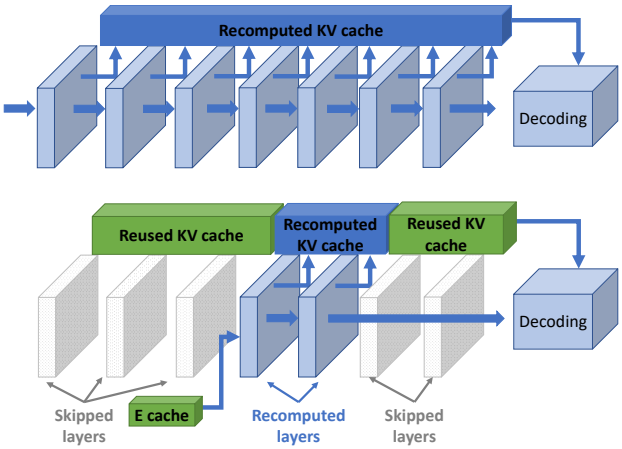

Figure 6: *DroidSpeak chooses the critical layer groups (layers 4-5) to re-compute, and reuse KV cache for other layers.*

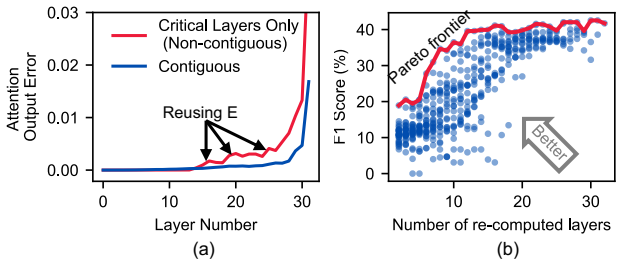

Figure 7: *(a) More transition points lead to higher output error. (b) An example result obtained by offline profiling. Each point represents the F1 score achieved when re-computing a specific group of contiguous layers. The Pareto frontier shows the maximum F1 score attainable for each number of re-computed layers.*

4 DroidSpeak Design

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 determine the layers to re-compute to reduce latency, while keeping the quality 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 quality. However, selecting all critical layers scattered across different parts of the LLM is suboptimal for both efficiency and quality.

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 for the whole context. While it is possible to obtain the E cache for layer l by performing a full prefill from the context starting from the first layer till layer l, this approach completely defeats the purpose of KV cache reuse.

To address this, we use the E cache from the sender model to start the recomputing at the layer when transitioning from KV cache reuse to recompute. We refer to this layer as a ***transition layer***. As depicted in Figure 6, for any transition layer (between reuse and recompute), the sender model must store and transmit the E cache to the receiver 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 Llama-3.1-70B since the KV cache size is optimized by group-query attention [7]. Thus, 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 loading KV cache alone.

The accuracy challenge: Furthermore, reusing the sender model's E cache at the transition layer also hurts the accuracy of the final output. This is because the E cache loaded from the sender model (starting point of the recomputation) already differs from the receiver 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 are often not appearing in contiguous groups (Figure 4), there will be multiple transition layers from reuse to recompute, introducing multiple deviations in E cache.

Figure 7(a) illustrates this. If we choose to recompute *only* critical layers (*i.e.,* layers 16–18, 20, and 25–27), we need to load E cache at layers 16, 20, and 25. However, whenever we load E cache, the error from E cache will be propagated to subsequent critical layers (*e.g.,* loading E cache at layer 16 populates errors to 16–18) and eventually to the output. Thus, even if all the critical layers are recomputed, this will lead to a substantial output error. In contrast, recomputing a contiguous group of layers from 16 to 27 avoids this problem by recomputing the KV cache of non-critical layers that are located between critical layers. As shown in Figure 7(a), re-computing a contiguous group of layers has much lower output error than only re-computing the critical layers.

4.2 Profiling for re-computation configuration

A key question still remains: how to determine the critical layer groups to be re-computed?

Based on Insight 3, the critical layers that are sensitive to KV cache differences vary little with the inputs. This motivates our design to choose the critical layer group based on some example inputs, and then apply to other new inputs. Specifically, for each model pair, we use a training dataset to determine the critical layer group. We refer to the critical layer group as the *re-computation configuration*. The goal is to understand the relationship between the critical layer groups to perform re-computation and its impact on generation quality.

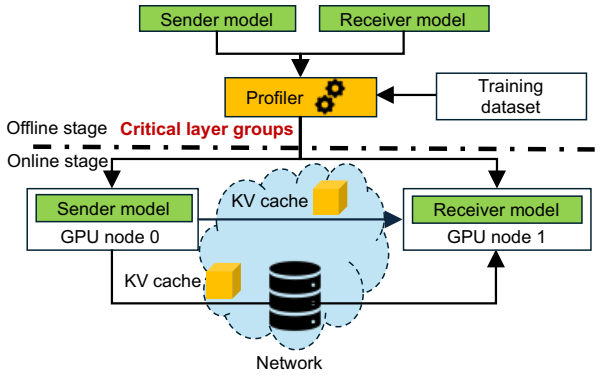

Figure 8: *Overall system design of DroidSpeak, including the offline stage (top) that profiles the critical layers of each pair of sender and receiver models (§4.2); and the online stage (bottom) that uses the profile to dynamically pick the critical layers and KV-cache loading strategy for each job (§4.3).*

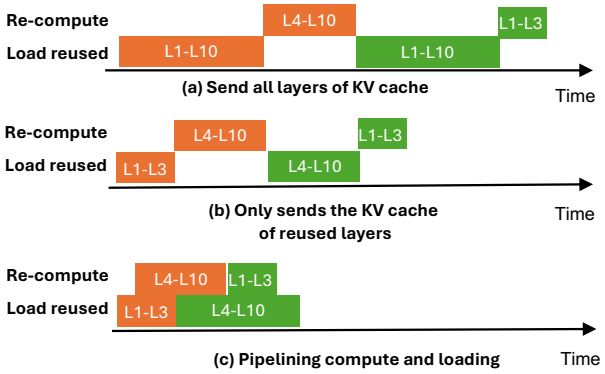

Figure 9: *Illustration of different KV cache transferring strategies. Each color represents the work (KV cache recomputation and loading reused KV caches) for the same query for one model.*

Figure 7(b) shows an example profiling result for the `glue_sst2` and `conllpp` model pair. Each point in the scatter plot represents the F1 score achieved for a specific number of re-computed layers. From these points, we obtain the Pareto frontier, which captures the highest F1 score achieved at a specific group of layers. For instance, a configuration with 11 re-computed layers is a good example on the Pareto frontier, since the F1 score drop is within 5% of the original accuracy of the receiver model, while the number of re-computed layers is minimized. Here, 5% is a hyperparameter that users can freely adjust according to their requirements.

Note that although the exact set of layers that need to be re-computed may differ between the training dataset and testing dataset, the key is that the Pareto frontier is similar across different datasets (§5.4).

Training data selection: We choose the training dataset to be similar to the domain of the task that the testing datasets target. In practice, we can use the fine-tuning dataset for profiling, as the inference requests of a specialized model should align with the data domain used to fine-tune it.

Profiling Overhead: The profiling overhead has a complexity of $O(l^2)$ where l is the number of layers in the LLMs. For example, for `Llama-3-8B` with 32 layers, it takes three hours on an A100 GPU. This one-time cost is negligible since these models are deployed at a very large scale [75, 111]. Furthermore, this cost can be substantially reduced by grouping layers when profiling. Profiling at the granularity of 2-layer groups, for instance, reduces the profiling time by about 3x. In the evaluation we show soon in §5, we use the 2-layer group profiling granularity.

4.3 DroidSpeak runtime design

As shown in Figure 8, DroidSpeak consists of two stages. The offline stage (as detailed in §4.2) uses a training dataset to profile the relationship between the value change of each layer on the generation quality. This step allows DroidSpeak to dynamically choose the right re-computation configuration based on available resources. Next, we detail the *online* stage that uses the offline profile to dynamically decide the critical layers of the KV cache and executes the partial recomputation on these critical layers to preserve high generation quality.

Specifically, in the online stage, DroidSpeak dynamically decides which point on the Pareto frontier should be used based on the latency SLO (details in Appendix§C). We assume the E cache and KV cache at each transition point are precomputed and stored, but they can also be generated by the sender model in real-time and sent directly from the sender model. Then, the receiver model selectively recomputes critical layer groups while reusing others, achieving a balance between computational efficiency and quality.

Although storing the E cache introduces extra GPU memory overhead on the sender model's side, only the E cache for *a single layer* – the transition layer – needs to be kept. This overhead is negligible compared to storing the KV cache across all layers by default. For example, in `Llama-3.1-8B-Instruct`, storing the KV cache for all layers for 10K tokens requires 1.2 GB of GPU memory, whereas storing the E cache for just one layer only needs 0.08 GB – about 6% of the total KV cache memory usage.

Smart KV Cache Loading: Since GPUs hosting different LLMs may reside in separate nodes, DroidSpeak needs to fetch KV cache from remote nodes frequently. Without careful design, this transfer can significantly increase the end-to-end latency, particularly when fewer layers are recomputed (*i.e.,* more KV cache layers need to be transferred).

Consider an example with two receiver models A and B. Each model has ten layers. Model A recomputes layers 4–10 and reuses KV cache for layers 1–3, while model B recomputes layers 1–3 and reuses KV cache for layers 4–10.

Figure 9 illustrates the timeline of three different loading and recomputation strategies. We assume that a request arrives at A at time=0, followed by a request to B after 2 time units, and that recomputing or transferring one layer (KV or E cache) takes 1 unit of time. The most naive approach is to load *all*

layers of the KV caches before recomputation begins in each job. For example, in model *A* (orange), this means first loading the E cache for layer 4 and the entire KV cache, followed by 7 units of re-computation time, similarly for *B*, resulting in a total TTFT of 47 (Figure 9(a)).

A slight improvement is to load only the reusable layers' KV cache for *A* and *B*, right before recomputation. This reduces the total TTFT to 30, as shown in Figure 9(b).

However, both approaches miss the opportunity to overlap recomputation and the loading of reused KV caches. Recomputation can start immediately after receiving the E cache of the transition layer. The optimal solution, shown in Figure 9(c), pipelines loading and recomputation. For model *A*, the E cache for layer 4 is transmitted first, enabling recomputation of layers 4–10 to start while the KV cache for layers 1–3 transfers in parallel. Similarly, for *B*, the loading of E/KV cache can begin before *A* finishes recomputation. This pipelined strategy reduces the total TTFT to 17—approximately a 2× improvement over the baseline in (b).

4.4 Implementation

We implement DroidSpeak with about 3K lines of code in Python, based on PyTorch v2.0, CUDA 12.0, and LMCache 0.1.4 [11, 25]. DroidSpeak operates the LLM inference serving engines through these interfaces:

- `store(Cache, context, LLM)`: We split the KV or E cache into layers, and store it in a key-value store in GPU memory.
- `fetch(context, LLM, layer_id) -> Cache`: Depending on what was stored previous `store()` call, this loads layer's KV or E cache of the corresponding `LLM`.
- `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 interfaces in vLLM [55] and LMCache [11]. 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 re-computation configuration from the offline profiling (§4.2), 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` [77] to fetch KV or E cache from a remote GPU node. All transmission will be placed on a CUDA Stream different from PyTorch's default computation stream [78], enabling us to overlap transmission of KV cache with recomputation and hiding the transmission delay.

5 Evaluation

The key takeaways from the evaluation are:

- Across three datasets and eight model pairs, DroidSpeak can reduce the prefill latency by 1.7–3.1× without compromising accuracy, with an average prefill speedup of 2.1×.
- In the online serving system, DroidSpeak achieves up to 4× improvement in throughput.
- DroidSpeak's profiling of recomputing layers is robust across different datasets and model types.

We also show more results in Appendix §B including comparison of DroidSpeak with smaller models, and applying DroidSpeak on math tasks.

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, Mistral-24B, Llama-3.1-8B, Llama-3-8B, Phi-3.5-mini-instruct, Llama-3-70B and Llama-3.1-70B, selected with the criteria in §3.1. These models are fine-tuned on the base foundation model for chat-enhancing tasks, coding tasks, and long context reasoning *et. al*. For Llama-3.1-70B and Llama-3-70B models, we use 4-bit quantized models with AWQ [61] to fit on one A100 GPU.

Note that the receiver models are not all directly fine-tuned from the sender model; they also include *two fine-tuned variants* derived from the same foundation model.

Hardware setting: We run the experiments on two A100 virtual machines connected with 200 Gbps NVIDIA Mellanox HDR InfiniBand links, namely `Standard_ND96amsr_A100_v4`, which contains 8 × 80GB A100 GPUs on each virtual machine.

Datasets: We evaluate DroidSpeak on six datasets, spanning three different tasks including *multi-hop question-answering, summarization, and code completion*, with their context length statistics summarized in Table 2. For each model pair, we report results on three of these datasets, following the criteria that the receiver model must achieve higher generation quality than the sender model on the chosen datasets.

Train/test split: As discussed in §4.2, DroidSpeak profiles the critical layer groups that has quality drop within 5% of the original quality with a training dataset offline. Specifically, we use 50 contexts from HotpotQA dataset as the training dataset to obtain the critical layer groups with the profiling mechanism mentioned in §4.2. We apply the critical layer groups on all the other testing datasets in the benchmark.

Quality metrics: We measure generation quality using the standard metric of each dataset, following prior work [12, 65, 100]. Specifically, we use F1 score for QA tasks (`hotpotQA`, `2wikimQA`, `multifieldQA_en`), which measures the probability that the generated answer matches the ground-truth answer for the question-answering task; Rouge-L score for summarization tasks (`multi_news`), which measures the longest

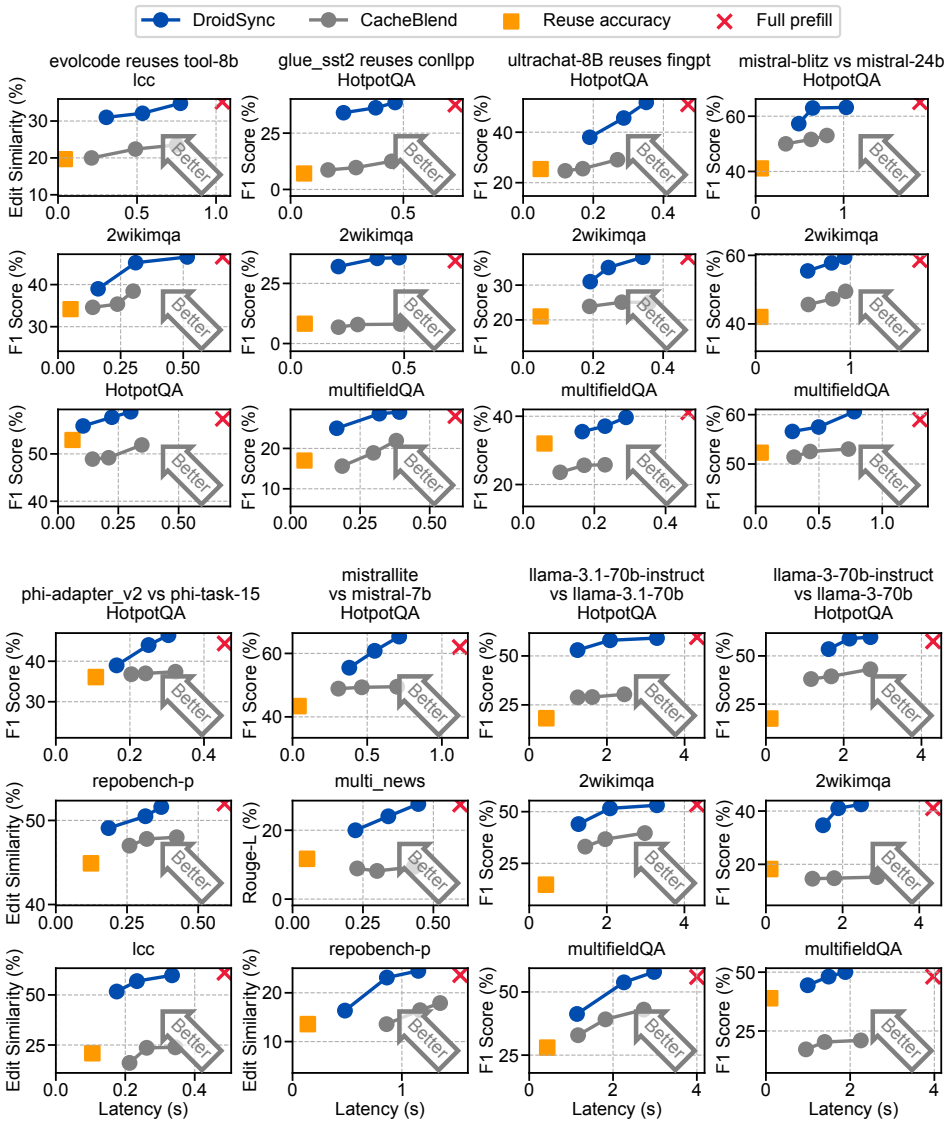

Figure 10: *Prefill delay and F1 score trade-off. DroidSpeak greatly reduces prefill latency while maintaining generation quality.*

common subsequence between the generated summarization and the ground-truth answer; and finally the code similarity score for code completion tasks (`lcc, repobench-p`), which measures the edit distance between the generated completed code and the ground-truth code.

System metrics: We use the system metrics listed in §2.1 to evaluate DroidSpeak compared with the baselines, including time-to-first-token (TTFT), time-between-tokens (TBT), end-to-end latency (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 with the following baselines:

- Full prefill: the receiver model prefills the text of the context with vLLM [55], representing the baseline of the highest computation overhead but the best quality we can get.
- Full KV cache reuse [34]: the receiver model directly reuses the KV cache from the sender model, and the receiver model runs decoding with the transferred KV cache.
- CacheBlend [100]: we extend CacheBlend's algorithm to determine the important tokens to re-compute for cross-LLM KV cache sharing, based on the difference between the re-computed KV cache and sender model's original KV cache for the first layer.
- Smaller models: In §B.2, 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.

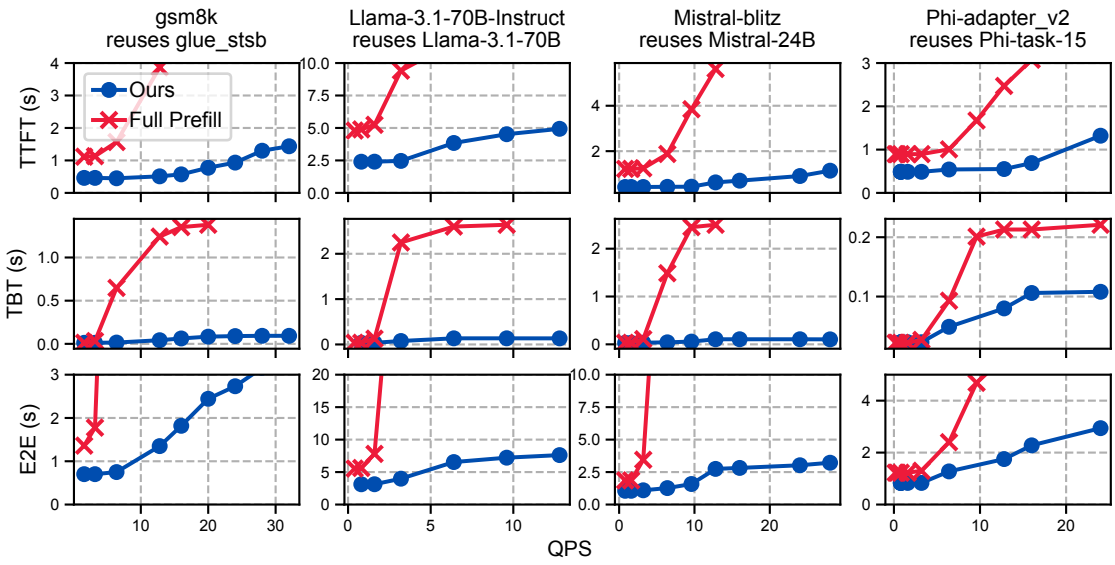

Figure 11: *The impact of arrival rate on TTFT, TBT, and E2E, when the DroidSpeak's quality is same as full prefill.*

5.2 Lower Latency with Preserved Accuracy

We first demonstrate DroidSpeak's reduction in prefill delay and accuracy trade-off in Figure 10. Across 8 pairs of models on three datasets, DroidSpeak achieves 1.7–3.1× 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. Compared to CacheBlend, DroidSpeak can achieve much better latency and quality trade-off. Specifically, DroidSpeak has 5–33% (average 16%) higher quality than CacheBlend at a similar prefill latency.

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 difference. DroidSpeak is better than CacheBlend in quality because DroidSpeak re-computes the critical layer groups that are most sensitive to KV cache deviation between models, while CacheBlend fails to spot the critical layers since it selects the tokens to re-compute based on the first layer.

5.3 Inference Throughput and Latency Improvement

To see the impact of DroidSpeak on improving the inference throughput[2] of an online LLM inference system, we emulated an online inference scenario by pairing the datasets with request arrival times following a Poisson distribution under different incoming rates to evaluate the performance of DroidSpeak in more practical workloads.

For the experiment, we deployed a Kubernetes cluster running vLLM Production Stack [90], using DroidSpeak 's customized Docker image. The cluster consisted of two nodes, each equipped with 8 A100 GPUs. We configured eight replicas for each model across these nodes. Model placement followed a simple strategy: for each model pair, we deployed four replicas of both the sender and receiver models on each node. Request routing was handled by vLLM Production Stack's round-robin algorithm, which splits incoming requests evenly across all model replicas.

As demonstrated in Figure 11, we compare the TTFT, TBT, and E2E impact under various request rates on the HotpotQA dataset. Due to the limit in space, we only show four pairs of models to illustrate. For DroidSpeak, we chose the configuration within 1% accuracy drop for these pairs of models.

TTFT: Since the full-recompute baseline has much 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 11.

Throughput: Assuming an SLO that avoids the effects of high queuing delays (knee of full prefill) on TTFT, TBT, and E2E latency, DroidSpeak can support 2-4× higher throughput. Although DroidSpeak can only reduce prefill computation, it becomes a bottleneck in the system, especially under long contexts, as long TTFTs cause the subsequent decoding to

[2]Here, inference request throughput refers to the number of requests the system can process per second, not network bandwidth throughput.

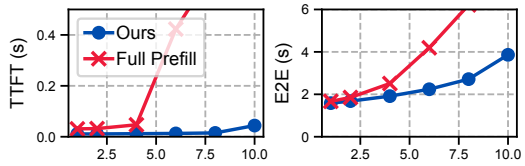

Figure 12: *Impact of the arrival rate on the TTFT and E2E delays of code agentic workflow.*

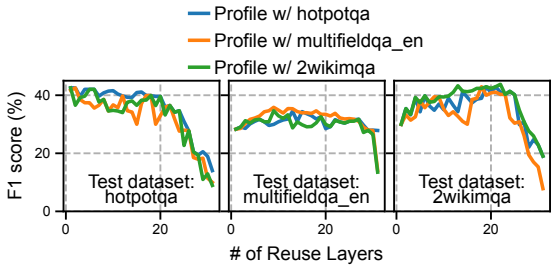

Figure 13: *Using the recompute layers profiled on training datasets works well on testing datasets.*

be queued for a long time. By reducing the TTFT bottleneck, DroidSpeak can thus increase the inference throughput.

5.4 Robustness across datasets

As discussed in §4.3, 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 13 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, with the average being 2 points. This result further validates the sufficiency and robustness of our profiling strategy.

Case study on coding agentic workflow: Next, we study how DroidSpeak performs under a real agentic workflow by orchestrating a coding agent system using MetaGPT [43], a state-of-the-art multi-agent framework. The system consists of two agents, a coder using evolcode model and a tester using tool-8b model. The coder is responsible for implementing Python functions according to the input prompt, and the tester is responsible for testing the coder's code and providing comments to the coder agent for the next round's modification.

We send the problems from the HumanEval dataset at various rates. In Figure 12, we plot the TTFT and E2E impact under different QPS, similar to the setup in §5.3, where DroidSpeak is plotted with the re-computation configuration that maintains the generation quality (pass@1 score of 52.5). DroidSpeak significantly improves the TTFT by 2.7× and brings down the E2E delay to finish the problem as well.

Mixture-of-Experts Model: We also evaluate DroidSpeak on a Mixture-of-Experts (MoE) model. As shown in Figure 14, where the sender model is Mixtral-8x7B and the receiver is Mixtral-8x7B-Instruct, DroidSpeak is able to achieve significant reductions in prefill latency as well. While MoE models may activate different experts for different inputs, this selection occurs only at the linear layers after the attention modules—where the KV cache is used in. Thus, DroidSpeak 's KV cache sharing remains effective for MoE architectures.

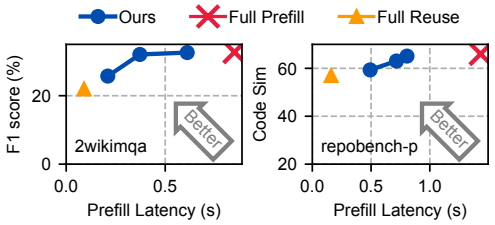

Figure 14: *Applying DroidSpeak on Mixtral-8x7B-Instruct sharing Mixtral-8x7B's KV cache, which are MoE models based on Mistral model architecture.*

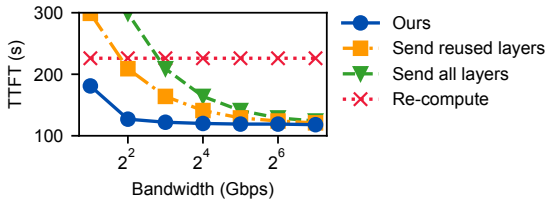

Figure 15: *Pipelining re-compute and loading the reused layers greatly reduces the total TTFT compared to the baselines of sequentially send and re-computes the KV cache.*

5.5 Impact of different network bandwidth

Figure 15 compares DroidSpeak and the baselines where the re-computation and loading of KV cache are not pipelined. We can see that DroidSpeak outperforms baselines under almost all bandwidth situations. Arguably, the absolute reduction in TTFT becomes smaller under high bandwidth, because the transmission delay becomes a smaller amount of the overall delay when the bandwidth is very high. We acknowledge that as networking speed increases, the advantage of loading KV caches in parallel with recomputation decreases compared to simply loading all or reused layers.

5.6 Profiling Overhead

Figure 16 shows the trade-off between profiling cost and generation quality. Profiling one layer at a time for the `glue_sst2` and `conllpp` model pair with 32 layers takes 3.6 hours, while grouping layers cuts this to 1 hour for 2-layer groups and 0.375 hours for 3-layer groups. The left side shows that coarser grouping maintains similar F1 scores and layer

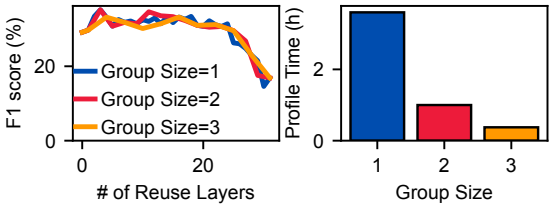

Figure 16: *DroidSpeak's profiling delay on glue_sst2 and conllpp model pair, on the multifieldQA dataset. Grouping the profiling layers at the granularity of multiple layers reduces profiling delay, without losing quality.*

reuse trade-off compared to per-layer profiling. We observe the same trend for the `evolcode` and `tool-8b` pair in §B.3.

6 Limitation and Future Work

KV cache sharing across different foundation models: DroidSpeak as-is only works for KV cache sharing across models originated from the same foundational model. Exploring the possibility and solution of KV cache sharing across different foundational models is a challenging problem that we leave for future work.

Re-computation adaptation with network bandwidth: In §4.3, we only consider adjusting the re-computation ratio based on system load. Future work may extend the adaptation algorithm to consider changes in network bandwidth, for example, expanding the range of critical layer groups when network bandwidth is limited.

Data drift in the re-computation configuration: DroidSpeak profiles the re-computation configuration for different models *offline*. We acknowledge that this profiling approach may result in quality degradation if the real test data drifts greatly from the training data used for profiling. To alleviate this issue, periodic re-profiling can be performed to update the re-computation configuration with newer data, which has closer distribution to the testing dataset. We leave this challenge to future work.

Applicability of DroidSpeak: Since DroidSpeak relies on the empirical insight that only a subset of layers are sensitive to KV cache reuse errors, it may not work for all model pairs, even those sharing the same foundation model. That said, we have validated that for model pairs in Section 5 and Section 3, DroidSpeak is able to significantly reduce prefill computation with little impact on generation quality.

KV cache reuse across multiple LLMs: Our current evaluation focuses on KV cache reuse between *two models*. Currently, KV cache reuse among *multiple models* needs to treat them as a collection of model pairs. We leave the optimizations of multi-model reuse to future work, such as maximizing shared layers across models with little impact in generation quality.

7 Related Work

Fine-tuning: Fine-tuning LLMs for specific tasks has gained importance, but it remains resource-intensive. Methods like parameter-efficient fine-tuning, including LoRA and LISA [72, 83, 93, 101] reduce the memory and computation needed for fine-tuning.

Multi-agent systems: Multi-agent systems show promise in areas such as coding [17, 42, 43, 49, 51, 79, 87], gaming [1, 18, 37, 63, 70, 108, 109], and social simulations [73, 74]. Fine-tuned LLMs as agents improve outcomes in question answering [16], tool learning [82], and personalization [59]. DroidSpeak reduces 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 [83, 93]. DroidSpeak is faster than them due to the elimination of prefill computation. Other works improve LLM serving including better scheduling [2, 55, 60, 67, 75, 84, 111], memory management for LoRA models [20, 83], and KV cache offloading [34, 45, 52, 56]. 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 [66, 81, 96]. 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 changes the number of recomputed layers.

KV cache reuse: Lots of prior works build efficient system for supporting KV cache reuse among different requests to accelerate inference [22, 23, 34, 45, 52, 80, 105], such as building multi-tier caching systems for KV cache based on request recency or importance. However, all of them assume KV cache is reused across different requests for *the same model*, while DroidSpeak targets at KV cache reuse across different models. Another line of research reduces the prefill delay when blending non-prefix KV caches from multiple contexts for the same model [35, 100]. Although CacheBlend also performs selective re-computation, DroidSpeak differs in both how and why the KV cache is re-computed.

8 Conclusion

In this work, we identified the core challenge of reducing repetitive computation in systems where multiple models work on a shared context. We presented DroidSpeak, a framework for KV cache sharing in compound AI systems. We identified only a subset of layers that require recomputation to maintain quality and show the robustness of our solution for a diverse range of model pairs, model types, and datasets.

9 Acknowledgment

We thank all the anonymous reviewers and our shepherd, Shay Vargaftik, for their insightful feedback and suggestions. Most of this project was done at Microsoft, and it is also funded by NSF CNS-2146496, CNS-2131826, CNS-2313190, CNS-1901466, UChicago CERES Center and two Google Faculty Awards.

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. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8038–8057, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, Santa Clara, CA, July 2024. USENIX Association.

[3] LangChain AI. Open deep research. https://github.com/langchain-ai/open_deep_research, 2025. GitHub repository, accessed September 15, 2025.

[4] Monica (Butterfly Effect AI). Manus ai agent. https://manus.im, 2025. Accessed: 2025-04-23.

[5] ai-dynamo. Dynamo: A datacenter scale distributed inference serving framework. https://github.com/ai-dynamo/dynamo, 2025. GitHub repository, release `v0.1.1`, accessed 21 April 2025.

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

[7] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[8] Anysphere Inc. Cursor: The ai code editor. https://www.cursor.com/, 2025. Accessed: 2025-04-23.

[9] Samuel Arcadinho, David Oliveira Aparicio, and Mariana S. C. Almeida. Automated test generation to evaluate tool-augmented LLMs as conversational AI agents. In Dieuwke Hupkes, Verna Dankers, Khuyagbaatar Batsuren, Amirhossein Kazemnejad, Christos Christodoulopoulos, Mario Giulianelli, and Ryan Cotterell, editors, *Proceedings of the 2nd GenBench Workshop on Generalisation (Benchmarking) in NLP*, pages 54–68, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

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

[11] LMCache Authors. Lmcache: A kv cache sharing layer for fast distributed llm serving. https://github.com/LMCache/LMCache, 2024.

[12] 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. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

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

[14] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. Distributed inference and fine-tuning of large language models over the internet. *Advances in neural information processing systems*, 36:12312–12331, 2023.

[15] 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.

[16] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

[17] Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, et al. Coder: Issue resolving with multi-agent and task graphs. *arXiv preprint arXiv:2406.01304*, 2024.

[18] Jiaqi Chen, Yuxian Jiang, Jiachen Lu, and Li Zhang. S-agent: self-organizing agents in open-ended environment. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

[19] 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.

[20] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024.

[21] 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. *arXiv preprint arXiv:2312.04333*, 2023.

[22] Shiyang Chen, Rain Jiang, Dezhi Yu, Jinlai Xu, Mengyuan Chao, Fanlong Meng, Chenyu Jiang, Wei Xu, and Hang Liu. Kvdirect: Distributed disaggregated llm inference. *arXiv preprint arXiv:2501.14743*, 2024.

[23] Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. IMPRESS: An Importance-Informed Multi-Tier prefix KV storage system for large language model inference. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 187–201, Santa Clara, CA, February 2025. USENIX Association.

[24] 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.

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

[26] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. A complete survey on llm-based ai chatbots. *arXiv preprint arXiv:2406.16937*, 2024.

[27] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, Singapore, December 2023. Association for Computational Linguistics.

[28] 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. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

[29] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint*, 2025.

[30] Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics.

[31] Jiabao Fang, Shen Gao, Pengjie Ren, Xiuying Chen, Suzan Verberne, and Zhaochun Ren. A multi-agent conversational recommender system. *arXiv preprint arXiv:2402.01135*, 2024.

[32] Shangbin Feng, Wenxuan Ding, Alisa Liu, Zifeng Wang, Weijia Shi, Yike Wang, Zejiang Shen, Xiaochuang Han, Hunter Lang, Chen-Yu Lee, et al. When one llm drools, multi-llm collaboration rules. *arXiv preprint arXiv:2502.04506*, 2025.

[33] Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R Costa-Jussà. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*, 2024.

[34] 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. In *Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC'24, USA, 2024. USENIX Association.

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

[36] GitHub. Github copilot. https://github.com/features/copilot, 2025. Accessed: 2025-04-23.

[37] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Yusuke Noda, Zane Durante, Zilong Zheng, Demetri Terzopoulos, Li Fei-Fei, Jianfeng Gao, and Hoi Vo. MindAgent: Emergent gaming interaction. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3154–3183, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[38] Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: a long-range pre-trained language model for code completion. In *Proceedings of*

the 40th International Conference on Machine Learning, ICML'23. JMLR.org, 2023.

[39] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: a survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024.

[40] Desta Haileselassie Hagos, Rick Battle, and Danda B. Rawat. Recent advances in generative ai and large language models: Current status, challenges, and perspectives. *IEEE Transactions on Artificial Intelligence*, 5(12):5873–5893, 2024.

[41] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[42] Samuel Holt, Max Ruiz Luyten, and Mihaela van der Schaar. L2MAC: Large language model automatic computer for extensive code generation. In *The Twelfth International Conference on Learning Representations*, 2024.

[43] 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.

[44] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290, 2024.

[45] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. Memserve: Context caching for disaggregated llm serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*, 2024.

[46] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[47] Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, et al. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025.

[48] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

[49] 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. *arXiv preprint arXiv:2312.13010*, 2023.

[50] Yuyang Huang, Yuhan Liu, Haryadi S Gunawi, Beibin Li, and Changho Hwang. Alchemist: Towards the design of efficient online continual learning system. *arXiv preprint arXiv:2503.01066*, 2025.

[51] Md. Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. MapCoder: Multi-agent code generation for competitive problem solving. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4912–4944, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[52] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Trans. Comput. Syst.*, September 2025. Just Accepted.

[53] Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Zhuoqing Mao. Compute or load KV cache? why not both? In *Forty-second International Conference on Machine Learning*, 2025.

[54] junyou li, Qin Zhang, Yangbin Yu, QIANG FU, and Deheng Ye. More agents is all you need. *Transactions on Machine Learning Research*, 2024.

[55] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[56] 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.

[57] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. Llm inference serving: Survey of recent advances and opportunities. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE, 2024.

[58] Haoran Li, Jinyu Wang, Yong Su, and Yue Zhang. Marft: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025.

[59] 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. *arXiv preprint arXiv:2401.05459*, 2024.

[60] 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.

[61] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 87–100, 2024.

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

[63] 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. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, page 1219–1228, Richland, SC, 2024. International Foundation for Autonomous Agents and Multiagent Systems.

[64] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code autocompletion systems. In *The Twelfth International Conference on Learning Representations*, 2024.

[65] 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. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 38–56, New York, NY, USA, 2024. Association for Computing Machinery.

[66] Zhenhua Liu, Zhiwei Hao, Kai Han, Yehui Tang, and Yunhe Wang. Ghostnetv3: Exploring the training strategies for compact models. *arXiv preprint arXiv:2404.11202*, 2024.

[67] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. Spotserve: Serving generative large language models on preemptible instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 1112–1127, New York, NY, USA, 2024. Association for Computing Machinery.

[68] Microsoft Docs. Azure openai service api version lifecycle. https://github.com/MicrosoftDocs/azure-ai-docs/blob/main/articles/ai-services/openai/api-version-deprecation.md, 2025. Accessed: 2025-04-23.

[69] Paul Mineiro. Online joint fine-tuning of multi-agent flows. *arXiv preprint arXiv:2406.04516*, 2024.

[70] Manuel Mosquera, Juan Sebastian Pinzón, Yesid Fonseca, Manuel Ríos, Nicanor Quijano, Luis Felipe Giraldo, and Rubén Manrique. Can llm-augmented autonomous agents cooperate? an evaluation of their cooperative capabilities through melting pot. *IEEE Transactions on Artificial Intelligence*, pages 1–10, 2025.

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

[72] 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. *Advances in Neural Information Processing Systems*, 37:57018–57049, 2024.

[73] 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.

[74] Joon Sung Park, Lindsay Popowski, Carrie Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Social simulacra: Creating populated prototypes for social computing systems. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery.

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

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

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

[78] PyTorch Team. torch.cuda.stream — pytorch 2.2 documentation. https://pytorch.org/docs/stable/generated/torch.cuda.Stream.html, 2024. Accessed: 2025-04-25.

[79] 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. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[80] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, Santa Clara, CA, February 2025. USENIX Association.

[81] Anthony Sarah, Sharath Nittur Sridhar, Maciej Szankin, and Sairam Sundaresan. Llama-nas: Efficient neural architecture search for large language models. In *European Conference on Computer Vision*, pages 67–74. Springer, 2024.

[82] 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. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[83] 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. Slora: Scalable serving of thousands of lora adapters. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 296–311, 2024.

[84] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 965–988, Santa Clara, CA, July 2024. USENIX Association.

[85] Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[86] Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. In *The Thirteenth International Conference on Learning Representations*, July 2025.

[87] 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.

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

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[90] vLLM Production Stack Team. vLLM Production Stack: reference system for k8s-native cluster-wide

deployment with community-driven performance optimization. https://github.com/vllm-project/production-stack, 2025. GitHub repository, release `vllm-stack-0.1.1`, accessed 21 April 2025.

[91] vLLM Project. AIBrix: Cost-efficient and pluggable infrastructure components for genai inference. https://github.com/vllm-project/aibrix, 2025. GitHub repository, release `v0.2.1`, accessed 21 April 2025.

[92] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

[93] 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.

[94] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[95] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

[96] Wenhan Xia, Hongxu Yin, and Niraj K. Jha. Efficient synthesis of compact deep neural networks: invited. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20. IEEE Press, 2020.

[97] Yifei Xia, Fangcheng Fu, Wentao Zhang, Jiawei Jiang, and Bin Cui. Efficient multi-task llm quantization and serving for multiple lora adapters. *Advances in Neural Information Processing Systems*, 37:63686–63714, 2024.

[98] Yingxuan Yang, Qiuying Peng, Jun Wang, Ying Wen, and Weinan Zhang. Llm-based multi-agent systems: Techniques and business perspectives. *arXiv preprint arXiv:2411.14033*, 2024.

[99] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[100] 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. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 94–109, New York, NY, USA, 2025. Association for Computing Machinery.

[101] 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. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1977–1992, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[102] Rui Ye, Xiangrui Liu, Qimin Wu, Xianghe Pang, Zhenfei Yin, Lei Bai, and Siheng Chen. X-mas: Towards building multi-agent systems with heterogeneous llms. *arXiv preprint arXiv:2505.16997*, 2025.

[103] Catherine Yeh, Yida Chen, Aoyu Wu, Cynthia Chen, Fernanda Viégas, and Martin Wattenberg. Attentionviz: A global view of transformer attention. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):262–272, January 2024.

[104] Yin Song and Chen Wu and Eden Duthie. amazon/MistralLite, 2023.

[105] Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 144–158, New York, NY, USA, 2025. Association for Computing Machinery.

[106] Shengbin Yue, Wei Chen, Siyuan Wang, Bingxuan Li, Chenchen Shen, Shujun Liu, Yuxuan Zhou, Yao Xiao, Song Yun, Xuanjing Huang, et al. Disc-lawllm: Fine-tuning large language models for intelligent legal services. *arXiv preprint arXiv:2309.11325*, 2023.

[107] Xiang Yue, Tianyu Zheng, Ge Zhang, and Wenhu Chen. MAmmoTH2: Scaling instructions from the web. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[108] 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.

[109] 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. In *ICLR*, 2024.

[110] Qingfei Zhao, Ruobing Wang, Yukuo Cen, Daren Zha, Shicheng Tan, Yuxiao Dong, and Jie Tang. LongRAG: A dual-perspective retrieval-augmented generation paradigm for long-context question answering. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22600–22632, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[111] 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. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, Santa Clara, CA, July 2024. USENIX Association.

A Detail statistics of the datasets

Table 2 shows the size of the datasets and the lengths of the contexts for these datasets.

Receiver Model	Sender Model
evolcode-3.1-8b	toolace-3.1-8b
glue_sst2	conllpp
mistral-blitz	mistral-24b
phi3.5-mini-instr-adapter-v2	phi-3.5-mini-instr-task15
llama-3-8b-sft-lora-ultrachat	fingpt-llama-3-8b
llama-3-70b-instruct	llama-3-70b
mistrallite	mistral-7b
llama-3.1-70b-instruct	llama-3.1-70b

Table 1: *The model pairs used in our paper. For each pair of models, we use the datasets that meet the requirements listed in §3.1 (i.e., the receiver model has better quality than the sender model).*

Dataset	Task	Size	Med.	Std.	P95
hotpotQA [99]	QA	300	10933	5160	18650
2wikimQA [41]	QA	200	7466	3976	10705
multifieldQA_en [12]	QA	150	8084	3849	14680
multi_news [30]	Summ.	200	7624	5923	17547
lcc [38]	Coding	200	12562	9220	26066
repobench-p [64]	Coding	200	14285	8665	30376

Table 2: *Size, context lengths, the tasks the dataset belongs to and evaluation metrics of datasets in the evaluation. Among the tasks, "QA" stands for question-answering, and "Summ." stands for summarization.*

B More results

B.1 Case study of other tasks and other models

Math task: To demonstrate that the mechanisms of DroidSpeak apply to other types of datasets, we apply DroidSpeak on a model pair where the receiver model is fine-tuned on math reasoning, and test on a math reasoning task.

In Figure 18(a), we run GSM8K [107] dataset on MAmmoTH2 [107]. We profile the re-computation configuration on the Math dataset [107]. Note that the Pareto frontier obtained follows a very similar pattern compared to the LongBench models and dataset, demonstrating the wide applicability of DroidSpeak.

B.2 Comparison against a smaller model

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

In Figure 18(b), we compare DroidSpeak on `Llama-3.1-70B-Instruct` and `Llama-3.1-8B-Instruct` in the HotpotQA dataset, 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`.

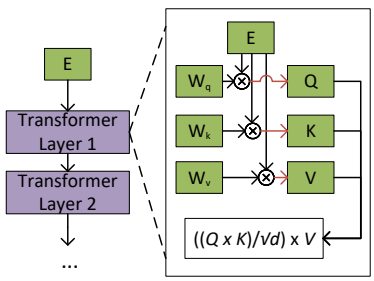

Figure 17: *Illustration of the use of embedding (E), query (Q), key (K), and value (V) tensors in self-attention in transformer-based LLMs.*

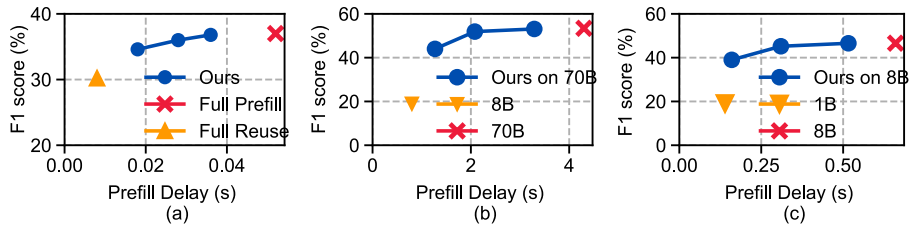

Figure 18: *(a) Prefill delay and accuracy trade-off for MAmmoTH2 (fine-tuned on math reasoning tasks). (b) DroidSpeak applied on Llama-3.1-70B-Instruct has higher accuracy than Llama-3-8B-Instruct. (c) DroidSpeak applied on Ultrachat-8B has higher accuracy on TinyLlama-1.1B-Chat.*

In Figure 18(c), we compare DroidSpeak on `Ultrachat-8B` and `TinyLlama-1.1B-Chat` on the 2wikimQA dataset, which is a smaller version of `Ultrachat-8B` while both fine-tuned on ultrachat_200k dataset [27]. Similary, we can see that compared with `Ultrachat-8B`, `TinyLlama-1.1B-Chat` reduces the prefill delay by 4.7×, while greatly reduces the F1 score. DroidSpeak, on the other hand, has much higher F1 score when applied to the `Ultrachat-8B` model.

One significant drawback of using a smaller model to achieve speedup is the overhead of switching between small and large models. For example, when additional resources become available, switching back to the larger model to improve serving quality incurs the overhead of loading the larger model back onto the GPU. In contrast, DroidSpeak can easily adapt to the available compute resources by adjusting the number of layers to be recomputed. This enables more possibilities for efficient scaling up or down on demand.

B.3 More results on profiling delay

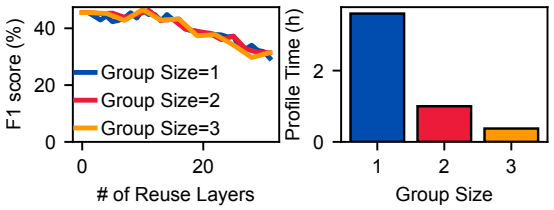

Figure 19: *DroidSpeak's profiling delay on evolcode and tool-8b model pair, on the 2wikimQA dataset. Grouping the profiling layers at the granularity of multiple layers reduces profiling delay, without losing quality.*

In Figure 16, we run the same experiment as in §5.6 on the `evolcode` and `tool-8b` model pair. We observe similar pattern as in §5.6 that profiling the recomputation configuration at a coarser granularity reduces profiling cost while maintaining generation quality.

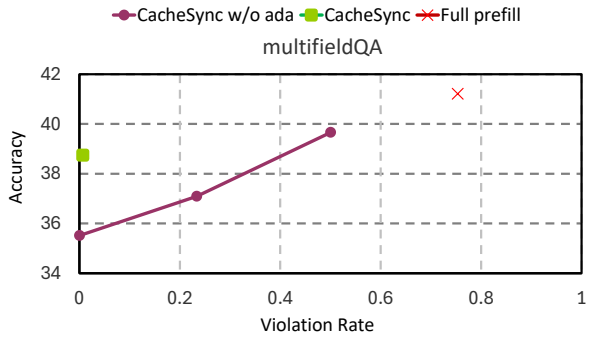

Figure 20: *DroidSpeak reduces SLO violation rate over DroidSpeak without adaptation and full prefill. Plotted with `ultrachat-8B` and `fingpt` model pair.*

C Dynamic Re-computation Configuration Adaptation

DroidSpeak initially checks the current workload intensity by monitoring the running and waiting requests in the vLLM engine [55]. If there are requests that are waiting to be executed, it indicates that the current workload is high and triggers an increase in the reuse ratio. In this case, DroidSpeak chooses a re-computation configuration that has the lowest number of re-computed layers above an accuracy target. When there are no queuing requests in the system, DroidSpeak estimates prefill latency based on the number of tokens of each request. The parameters for estimation can be obtained through the profiling phase. DroidSpeak then selects the highest recomputation ratio satisfying the latency SLO for each request to maximize accuracy.