DiSCo: Device-Server Collaborative LLM-Based Text Streaming Services

Ting Sun[1], **Penghan Wang**[2], **Fan Lai**[1]*
[1] University of Illinois Urbana-Champaign, United States
[2] Purdue University, United States
suntcrick@gmail.com, wangpenghan381@gmail.com, fanlai@illinois.edu

Abstract

The rapid rise of large language models (LLMs) in text streaming services has introduced significant cost and Quality of Experience (QoE) challenges in serving millions of daily requests, especially in meeting Time-To-First-Token (TTFT) and Time-Between-Token (TBT) requirements for real-time interactions. Our real-world measurements show that both server-based and on-device deployments struggle to meet diverse QoE demands: server deployments face high costs and last-hop issues (e.g., Internet latency and dynamics), while on-device LLM inference is constrained by resources.

We introduce *DiSCo*, a device-server cooperative scheduler designed to optimize users' QoE by adaptively routing requests and migrating response generation between endpoints while maintaining cost constraints. *DiSCo* employs cost-aware scheduling, leveraging the predictable speed of on-device LLM inference with the flexible capacity of server-based inference to dispatch requests on the fly, while introducing a token-level migration mechanism to ensure consistent token delivery during migration. Evaluations on real-world workloads—including commercial services like OpenAI GPT and DeepSeek, and open-source deployments such as LLaMA3—show that *DiSCo* can improve users' QoE by reducing tail TTFT (11-52%) and mean TTFT (6-78%) across different model-device configurations, while dramatically reducing serving costs by up to 84% through its migration mechanism while maintaining comparable QoE levels.

1 Introduction

Large language models (LLMs) have revolutionized various applications, with over 60% focusing on conversational interactions such as chatbots (Grand View Research, 2023). Meeting high serving demands requires scaling deployments across on-premise servers in the cloud and on-device inference, as seen in Apple Intelligence (Gunter et al., 2024) and Google's Gemini Nano (Google, 2024). The Quality of Experience (QoE) for interactive applications is primarily evaluated by two critical metrics: Time-To-First-Token (TTFT) in the prefill stage, which quantifies the initial response latency, and Time-Between-Token (TBT) during the decode stage, which measures the consistency of token delivery speed (Databricks, 2023; Liu et al., 2024a,c).

On-server deployments lower serving costs by sharing infrastructure among many requests but often introduce unpredictable high latency due to request queuing delays (Agrawal et al., 2024) and the internet speed fluctuations. While on-device deployment is able to serve increasingly capable LLMs with sufficient accuracy, it suffers from slow processing speeds for long prompts and high energy consumption. For example, a fully-charged iPhone can only operate for less than two hours running an LLM with 7B parameters (Liu et al., 2024d).

This paper introduces a novel paradigm for cost-constrained device-server cooperative inference. We incorporate both server usage (e.g., monetary costs) and device energy costs via a dynamic exchange rate, which can be adjusted by endpoint users to balance response generation between the cloud and devices. As such, we can strategically distribute inference requests between endpoints and dynamically migrate ongoing token generation to maximize QoE. However, realizing this vision presents several fundamental challenges:

- **Unified Cost Management:** Total serving costs include resource expenditures from both endpoints—monetary costs from server API usage and energy costs from device computation. The relative value of energy costs varies dynamically based on device context (e.g., battery level, charging status) and user preferences for server spending, making it challenging to establish a unified optimization strategy.

* Corresponding author

Findings of the Association for Computational Linguistics: ACL 2025, pages 14259–14277

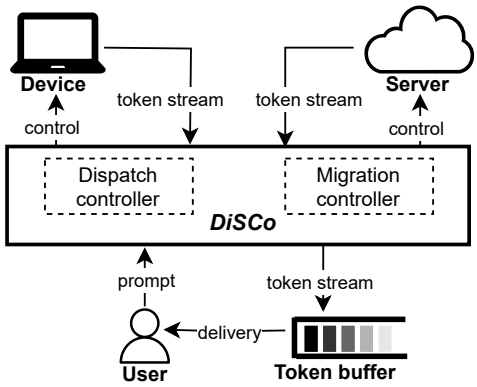


Figure 1: *DiSCo* acts as a middleware to optimize QoE by adaptively dispatching and migrating response generation between device and server endpoints under cost constraints.

- **Runtime Uncertainty:** The dynamic nature of networks (e.g., latency jitters) and serving loads make it challenging to accurately predict TTFT for in-flight request migration. Moreover, any scheduling mechanism must be lightweight to avoid introducing large overhead to the already latency-sensitive services.
- **Migration Impact on Token Delivery:** While dynamic migration between endpoints can reduce overall operating costs, it risks disrupting TBT. The challenge lies in determining when and how to migrate while minimizing user experience degradation and cost increase.

As shown in Figure 1, we introduce *DiSCo*, a Device-Server Cooperative scheduler that addresses these challenges via two key innovations:

- **Cost-Aware Dispatching Policies:** We introduce two dispatching mechanisms targeting different cost constraints. For server cost constraints, we employ a length-threshold based dispatching mechanism that routes requests shorter than a dynamically computed threshold to devices. For device energy constraints, we implement a delay-based dispatching mechanism where devices wait for a computed interval before starting local inference. Both mechanisms adapt their thresholds based on unified cost measures that combine server monetary costs and device energy consumption.
- **Token-Level Migration Framework:** We enable seamless generation handoff between endpoints through a novel migration protocol that preserves the consistency of token delivery. Our framework employs delayed migration timing to minimize interruption, while a token buffer ensures smooth delivery during transitions. This design maintains user experience while saving resource costs across endpoints.

Through extensive evaluation using real-world traces from commercial LLM streaming API services, including GPT and DeepSeek, and on-device deployments, we demonstrate that *DiSCo* improves mean and tail TTFT by up to 50% without TBT violation, significantly reducing costs.

Overall, we make the following contributions:

- We characterize QoE challenges in device-server cooperative LLM inference through extensive real-world measurements.
- We design novel scheduling policies that optimize QoE under cost constraints.
- We develop a token-level migration framework to enable generation handoff between endpoints, preserving token delivery consistency.
- We demonstrate *DiSCo*'s effectiveness in commercial services and open-source benchmarks.

2 Background and Motivation

2.1 LLM Token Mixture and Routing

Device-server collaborative approaches have evolved in two directions. First, systems like EdgeShard (Zhang et al., 2024) and WDMoE (Xue et al., 2024a) partition LLMs across multiple endpoints when a single device cannot host the entire model. LLMCad (Xu et al., 2023) uses on-device models to reduce server costs, while PerLLM (Yang et al., 2024) optimizes energy consumption across devices and servers under constraints. Second, routing-based approaches (Ong et al., 2024; Ding et al., 2024) balance cost and accuracy by directing simple requests to small models and complex queries to advanced ones. However, these approaches do not optimize token delivery metrics (TTFT and TBT) under cost constraints.

2.2 LLM-Based Text Streaming Applications

Over 60% of LLM-backed applications focus on streaming conversational interactions, such as chatbots, virtual assistants, and language translation. QoE in these text streaming services is often quantified by two critical metrics: time-to-first-token (TTFT) for *initial responsiveness* and time-between-tokens (TBT) for *delivery smoothness* throughout the entire interaction timeline.

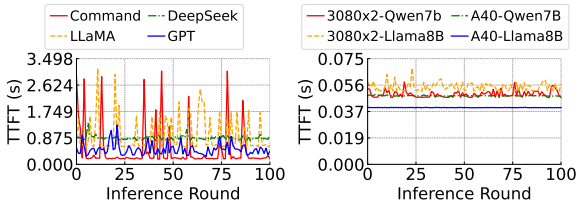
Current LLM systems struggle to meet user expectations for these metrics, with TTFTs ranging

from hundreds of milliseconds to over ten seconds—far exceeding the ideal latencies of tens of milliseconds for interactive applications (Mäki-Patola and Hämäläinen, 2004; Žádník et al., 2022). Token consumption patterns vary by output modality: In visual text scenarios, reading speeds differ across demographic groups, with the majority (52%) aged 25-44 reading 4-5 tokens per second, while older groups generally read more slowly (Liu et al., 2024a; Brysbaert, 2019; Petrov et al., 2024). Audio output consumption shows more consistency, averaging 3-4 tokens per second across languages (Liu et al., 2024a; Parachuk, 2022; Barnard, Dom, 2022). Notably, conventional evaluation metrics like token generation throughput or average time-per-output-token provide incomplete insights, as they fail to capture the crucial relationship between token delivery timing and actual user consumption patterns.

2.3 Limitations of Existing Text Streaming Applications

Existing LLM serving primarily relies on two deployment paradigms: on-device and on-server inference. With rapid hardware and software advancements, on-device LLMs have achieved sufficient accuracy levels for many applications (Phan et al., 2025), as evidenced by the integration of Apple Intelligence (Gunter et al., 2024) and Google's Gemini Nano (Google, 2024) into iOS and Android platforms, where they effectively handle text completion and message composition tasks. While on-device LLMs may still be inadequate for complex tasks (e.g., advanced mathematical reasoning), we focus on the growing category of applications where current on-device models already achieve satisfactory accuracy. For these applications, the challenge is not model capability, but rather the substantial monetary or energy cost demands of LLM inference.

Unfortunately, both serving paradigms face challenges. On-device inference, though enabling faster generation powered by its dedicated resources (Song et al., 2023; Xue et al., 2024b), suffers from extended TTFT for long prompts due to limited processing speeds and substantial energy consumption that scales linearly with response lengths (Li et al., 2024b). For instance, a fully-charged iPhone running a 7B parameter LLM can only operate for less than two hours (Liu et al., 2024d)—insufficient for day-long mobile use.



(a) On-Server TTFTs. (b) On-Device TTFTs.

Figure 2: On-device TTFT performance is more stable.

On the other hand, on-server deployments require request batching to amortize costs due to the high resource demands, but this introduces issues like queuing delays, resource contention from batching (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024), and last-hop network latency variations (Li et al., 2024a). Our measurements reveal that these factors can cause significant TTFT spikes for GPT-4-mini, from 0.3 seconds to several seconds during high-load periods.

Given these complementary limitations, we investigate the following research question: *Can a cooperative paradigm be designed to combine on-server and on-device inference to improve QoE while managing both energy and monetary costs?*

3 Characterizing LLM Inference

This section characterizes the LLM inference performance of on-server and on-device paradigms, which informs our design.

We evaluate four commercial streaming LLM APIs: OpenAI's GPT-4o-mini (OpenAI, 2024), DeepSeek's DeepSeek-V2.5 (DeepSeek, 2024), Cohere's Command (Cohere, 2024), and Hyperbolic-hosted LLaMA-3-70b-Instruct (Hyperbolic, 2024). For on-device analysis, we deploy Qwen-2.5-7B-Instruct (Alibaba, 2024) and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) on both server-grade (NVIDIA A40, 48GB) and consumer-grade (dual NVIDIA RTX 3080, denoted as 3080x2) GPUs. We sample 1,000 requests from the Alpaca dataset (Taori et al., 2023), following a Poisson distribution with a mean request arrival interval of 30 seconds.

TTFT characteristics. Our measurements reveal the contrasting TTFT patterns between on-device and on-server inference. As shown in Figure 2, on-device inference exhibits stable TTFTs when processing identical prompts at 60-second intervals, primarily reflecting the prefill duration due

Model	Deployment	Pearson Coef.
Command	Server	0.0142
GPT-4o-mini	Server	0.0236
DeepSeek-V2.5	Server	-0.0273
LLaMA-3-70b-Instruct	Server	0.0402
LLaMA-3.1-8b-Instruct	Device	0.8424

Table 1: Pearson coefficient between prompt length and TTFT in on-server deployment is weak.

to dedicated local hardware resources. In contrast, on-server inference experiences high variations and significant tail latency, attributed to network delays, request queuing, and resource contention.

Unlike previous works that focus solely on prefill latency (e.g., (Gim et al., 2024; Kamahori et al., 2024)) or the sum of queuing and prefill (e.g., (Agrawal et al., 2024; Qin et al., 2024)), we measure user-perceived TTFT as a comprehensive sum of network, queuing, and prefill latencies. Our extensive evaluations across four major LLM providers consistently demonstrate this improvement. This methodology aligns with industry benchmarks, such as Artificial Analysis (Artificial Analysis, 2025)'s continuous monitoring of diverse LLM services. Notably, since most on-device LLMs struggle with prefilling or understanding long prompts, we focus on short prompts, where request queuing delay and batch completion waiting time during generation dominate the overall TTFT.

We summarize the TTFT performance of 1,000 requests in Table 1. We observe that on-device TTFT scales linearly with prompt length due to hardware constraints (Li et al., 2024b), while on-server TTFT shows minimal prompt-length sensitivity through advanced resource scaling (Zhong et al., 2024; Patel et al., 2024; Hu et al., 2024). Note that LLM providers (e.g., Microsoft Azure (Microsoft Azure, 2025), DeepSeek (DeepSeek, 2025), Together.ai (Together AI, 2025), Hyperbolic Labs (Hyperbolic, 2025)) typically do not offer explicit TTFT SLOs, likely due to the high cost and complexity of maintaining such guarantees across diverse models and prompts.

TBT characteristics. TBT characterizes the I/O-bound decode stage latency. Analysis of temporal samples and distributions across six setups (Figure 3) reveals higher TBT variability in on-server inference compared to on-device execution. Moreover, both deployments achieve generation speeds exceeding user consumption rates (§ 2.2), making cooperative serving practical.

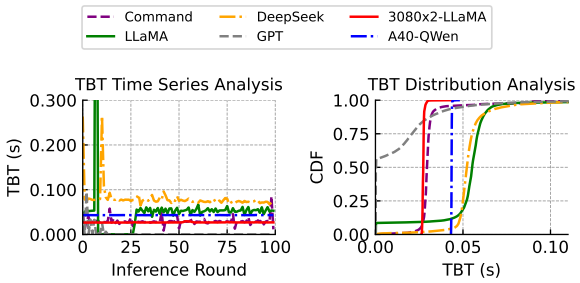


Figure 3: On-device TBT performance is more stable. [1]

Opportunities and challenges. Our studies further reveal that as on-device models continue to improve—often fine-tuned for specific tasks (Gunter et al., 2024; Liu et al., 2024d)—they are progressively achieving performance parity with on-server models in popular applications like instruction-following and translation (detailed in §5 and Appendix D). However, deploying these models on-device introduces challenges such as long prefill latency and startup overhead.

On the other hand, our real-world studies of conversational workloads highlight key opportunities: (i) on-server TTFT is largely unpredictable and shows minimal correlation with prompt length, whereas on-device TTFT scales nearly linearly with prompt length and is highly predictable; and (ii) both paradigms achieve token generation speeds that exceed typical user consumption rates.

By integrating these findings—especially the predictable performance of on-device inference and the elastic scaling capabilities of server-based inference—we observe opportunities for optimization in cost-constrained device-server cooperative serving. Dynamic request migration between server and device endpoints during response generation can yield significant cost savings.

4 *DiSCo* Policies

DiSCo optimizes both QoE and cost through (1) dispatch control that determines where to initiate token generation, and (2) migration control that enables dynamic handoff during generation. The dispatch controller optimizes TTFT by strategically routing requests, while the migration controller maintains consistent TBT while reducing costs.

[1] On-server inference, such as in GPT, streams tokens with each packet containing multiple tokens, resulting in near-zero perceived TBTs.

4.1 Problem Formulation

We propose a unified cost model combining both monetary bills from on-server inference and energy bills from on-device inference. Let c_s^p and c_s^d denote the per-token monetary costs for server prefill and decode phases, respectively, while c_d^p and c_d^d represent the per-token energy costs for device prefill and decode phases. Integration of energy and monetary costs is done by a dynamic exchange rate λ, adjusted by users to reflect their preferences. We offer a user-friendly tunable budget ratio $b \in [0, 1]$, representing the additional cost allowance beyond baseline costs. Our optimization objectives focus on: (1) minimizing both mean and tail TTFT, and (2) maintaining consistent token delivery at a specified pace (i.e., stable TBT).

4.2 Dispatch Controller: Cost-Aware Request Routing

Based on our analysis in §3, server-side TTFT shows weak correlation with prompt length due to various factors (network delay, request queuing, etc.). We model server TTFT as a known distribution, obtained either from server-provided information or device-side profiling. In contrast, device-side TTFT exhibits a linear relationship with prompt length, with the coefficient determined through offline profiling.

Our key insight is that the optimization problem naturally decomposes into two scenarios based on dominant cost factors: device-constrained scenarios where energy consumption is the primary bottleneck, and server-constrained scenarios where API monetary costs dominate. This decomposition enables efficient solutions. The pseudocode for the dispatch controller is attached in Appendix F.

Device-Constrained Optimization. When device costs dominate ($\min(c_d^p, c_d^d) > \max(c_s^p, c_s^d)$), we need to carefully manage device resource usage under a budget constraint $\mathbb{E}[I_d(l)l] \leq b \cdot \mathbb{E}[l]$, where l is the prompt length and $I_d(l)$ indicates device execution. The key challenge is balancing between two goals: leveraging device execution to bound worst-case latency while conserving energy on shorter prompts when possible.

Our solution uses a waiting-time strategy: for each prompt of length l, we first try server execution and wait for time $w(l)$ before potentially starting device execution. This conserves device energy when the server responds quickly. We determine the optimal wait time through a two-phase approach:

- **Phase 1 (Tail Protection):** We reserve a budget portion α for worst-case scenarios by setting a maximum wait time $w_{tail} = F^{-1}(1 - \min(\alpha, b))$, where $F(\cdot)$ is the server TTFT distribution. This ensures we have device resources ready when server latency exceeds its $(1 - \min(\alpha, b))$-th percentile.
- **Phase 2 (Average Case):** With the remaining budget $(b - \alpha)$, we set length-dependent wait times:

$$w(l) = \begin{cases} 0 & \text{if } l \leq l_{th} \\ \min(\beta l, w_{tail}) & \text{otherwise} \end{cases} \quad (1)$$

  where l_{th} is a threshold below which we start device execution immediately, and β is chosen to satisfy:

$$\int_{l_{th}}^{\infty} (1 - F(\beta l)) \cdot c_d^p \cdot l \cdot p(l) dl = (b - \alpha) \cdot \mathbb{E}[l] \quad (2)$$

This design guarantees worst-case TTFT through w_{tail} while optimizing average performance by adaptively adjusting wait times based on prompt length. Whichever endpoint (server or device) generates the first token continues to the decode phase, while the other terminates.

Server-Constrained Optimization. When server costs dominate ($\max(c_s^p, c_s^d) > \min(c_d^p, c_d^d)$), we need to carefully manage server resource usage under a budget constraint $\mathbb{E}[I_s(l)l] \leq b \cdot \mathbb{E}[l]$, where $I_s(l)$ indicates server execution. Our analysis in §3 shows that device TTFT scales linearly with prompt length as $T_d(l) = kl + c$, while server TTFT has minimal length correlation. This suggests a length-based routing strategy: short prompts run on the device to conserve server budget, while long prompts use both endpoints to minimize TTFT.

We determine the length threshold l_{th} by:

$$\int_0^{l_{th}} l \cdot p(l) dl = (1 - b) \cdot \mathbb{E}[l] \quad (3)$$

This ensures prompts shorter than l_{th} consume exactly $(1 - b)$ fraction of total expected tokens through device-only execution, leaving the remaining longer prompts with sufficient server budget for concurrent execution on both endpoints.

4.3 Migration Controller: Cost-Efficient Token Delivery

When both endpoints process a request, the constrained endpoint may win the prefill phase but incur higher decode costs. In such cases, we can migrate token generation to the other endpoint to reduce total cost while maintaining quality.

Theoretical Migration Framework. The token-level migration protocol ensures seamless handoffs by leveraging the gap between token generation (r_g) and consumption (r_c) rates. The migration trigger is determined by comparing the cost savings against migration overhead:

$$C_{\text{migration}} = \Delta c_{\text{decode}} \cdot l_{\text{remaining}} > \text{Overhead}_{\text{migration}} \tag{4}$$

where Δc_{decode} is the per-token cost difference between endpoints and $l_{\text{remaining}}$ represents the expected remaining sequence length. This formulation ensures that migration only occurs when the projected cost savings exceed the overhead of transferring control between endpoints.

Efficient Token Transfer. When endpoints share the same vocabulary, we transmit token IDs rather than complete token representations. Empirical analysis using the Alpaca dataset (Taori et al., 2023) with the cl100k_base tokenizer (OpenAI, 2022) (used by GPT-3.5/4) demonstrates that token ID transmission versus UTF-8 encoded text yields 35.62% reduction in data volume when using minimum byte encoding (3 bytes per token), and 54.40% reduction in data volume when using minimum bit encoding (17 bits per token). These efficiency gains are particularly valuable in bandwidth-constrained environments, helping to minimize latency during migration.

Additionally, we avoid transferring intermediate states (e.g., attention key-value cache) for two practical reasons: (1) endpoints often employ different model architectures optimized for their respective hardware, making state transfer incompatible, and (2) intermediate state transfer would incur significant network overhead. For models with different vocabularies, we first convert tokens to text before re-tokenizing on the target model to ensure semantic consistency.

Migration is triggered when the projected cost savings exceed overhead:

$$C_{migration} = \Delta c^d_{decode} \times l_{remaining} \tag{5}$$

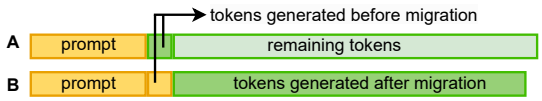


Figure 4: Token generation migration between endpoints. Row A shows the original sequence on the source endpoint, while Row B shows the sequence after migration to the target endpoint, maintaining consistent token delivery while reducing cost.

where $\Delta c^d_{decode} = |c^d_s - c^d_d|$ and $l_{remaining}$ denote the per-token decode cost difference between endpoints, and the expected remaining sequence length, respectively.

Buffer-Based Migration Protocol. To ensure smooth token delivery during migration, we introduce a token buffer that leverages the natural gap between token generation speed (r_g tokens/s) and human consumption rate (r_c tokens/s, typically $r_g > r_c$). The buffer size is set to:

$$B = r_c \times t_m \tag{6}$$

where t_m is the estimated migration overhead time. Migration begins only when the buffer contains enough tokens (B) to overshadow the migration latency. Importantly, the source endpoint stops generating new tokens once the buffer is filled, preventing any potential conflicts or branching during the transition. This ensures deterministic token delivery as the target endpoint takes over generation.

As shown in Figure 4, this design enables seamless handoff: the source endpoint (Row A) continues generation until the buffer is filled, then stops to allow the target endpoint (Row B) to take over, ensuring uninterrupted token delivery to users despite the underlying endpoint transition.

5 Evaluation

Through extensive experimentation with four production-grade LLM services and state-of-the-art open-source models, we demonstrate *DiSCo*'s exceptional performance. Our rigorous evaluation spanning diverse deployment scenarios reveals that *DiSCo* delivers remarkable improvements, reducing both mean TTFT (6-78%) and tail TTFT (11-52%) while achieving cost savings of up to 83.6%.

5.1 Evaluation Setup

Testbeds and Workloads. Our testbed is a server with 4 NVIDIA A40 GPUs, each with 48GB of memory. We evaluate *DiSCo* using both commercial LLM traces and on-device deployments. For

server-side evaluation, we collect traces from four production services: OpenAI's GPT-4o-mini (OpenAI, 2024), DeepSeek-V2.5 (DeepSeek, 2024), Cohere's Command (Cohere, 2024), and Hyperbolic-hosted LLaMA-3-70b-Instruct (Hyperbolic, 2024). For on-device evaluation, we test three representative device-model configurations (Li et al., 2024b): Pixel 7 Pro with Bloom 1.1B (31.32/13.93 tokens/s for prefill/decode), Pixel 7 Pro with Bloom 560M (51.80/20.14 tokens/s), and Xiaomi 14 with Qwen 1.5 0.5B (79.90/21.47 tokens/s). These configurations span different compute-capability trade-offs in mobile environments. For end-to-end cost comparison, we quantify server costs using commercial API token pricing and device costs using FLOPs-based energy consumption. The detailed cost analysis can be found in Appendix E.

Baselines. We compare *DiSCo* with four on-server, on-device, and cooperative deployments:

- *vLLM* (Kwon et al., 2023): Processes all requests using remote server-based deployment.
- *llama.cpp* (Gerganov, 2024): Processes all requests using local device-based deployment.
- *Stoch-S*: A server-constrained approach that randomly routes requests to the device while capping the server budget.
- *Stoch-D*: A device-constrained approach that randomly routes requests to the server while capping the device budget.

For end-to-end cost comparison, we include two additional baselines: *DiSCo-D w/o Migration* and *DiSCo-S w/o Migration*.

Metrics. We evaluate the system performance using both TTFT and TBT, including their mean and tail values. They are analyzed across varying cost budgets, defined as the ratio of input tokens processed by the constrained endpoint (device or server) to the total input tokens. For each experiment, we report the mean value over 10 runs.

5.2 End-to-end Performance

***DiSCo* improves TTFT performance.** Figure 6 and Table 2 show that *DiSCo* significantly outperforms baseline methods in both device- and server-constrained settings, showing improvements across mean and tail (P99) TTFT metrics for various services, including GPT, LLaMA, DeepSeek, and Command. In the GPT experiments, *DiSCo* demonstrates particularly notable tail latency reductions, decreasing P99 TTFT by up to 40% relative

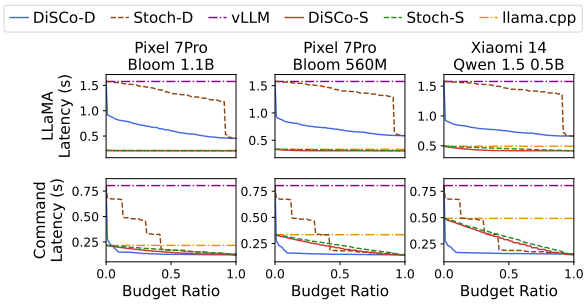


Figure 5: Mean TTFT reduction of *DiSCo* remains significant on DiffusionDB trace.

Platform	Constraint	Tail TTFT Reduction		
		Pixel 7Pro B-1.1B	Pixel 7Pro B-560M	Xiaomi 14 Q-0.5B
GPT	Server	23.85%	37.41%	44.04%
	Device	26.39%	21.48%	16.32%
LLaMA	Server	11.08%	23.09%	26.29%
	Device	35.67%	29.30%	21.29%
DeepSeek	Server	0.00%*	3.88%	15.53%
	Device	30.91%	28.01%	25.08%
Command	Server	47.93%	50.93%	52.23%
	Device	34.78%	31.53%	24.42%

Table 2: Average reduction of tail TTFT compared to stochastic dispatching across the whole cost budget range. Devices include Pixel 7 Pro and Xiaomi 14, while models include Bloom-1.1B, Bloom-560M, and Qwen-1.5-0.5B. (*Tail TTFT remains constant.)

to Stochastic dispatching across all device configurations, while mean TTFT is also reduced substantially, with reductions between 20-30% across diverse budget ratios. In the LLaMA setup, we observe a unique trade-off pattern. For budget ratios below 20% when the device is the constrained endpoint, *DiSCo* exhibits a slightly higher mean TTFT than the baseline. This outcome is intentional, as *DiSCo* prioritizes tail latency reduction in low-budget scenarios, yielding substantial gains in P99 TTFT—reducing tail latency by up to 50%.

DeepSeek and Command experiments demonstrate similar patterns of improvement as the previous two traces, with *DiSCo* consistently outperforming baseline approaches. In the DeepSeek scenario, *DiSCo* maintains stable latency even as the budget ratio increases, whereas the baseline systems show increasing latency variance.

***DiSCo* retains TBT performance while lowering the cost.** Table 3 evaluates *DiSCo*'s TBT performance across various traces under both server and device constraints. For requests involving migration, we measure two key metrics: the average number of migrations per request and the tail (P99) TBT

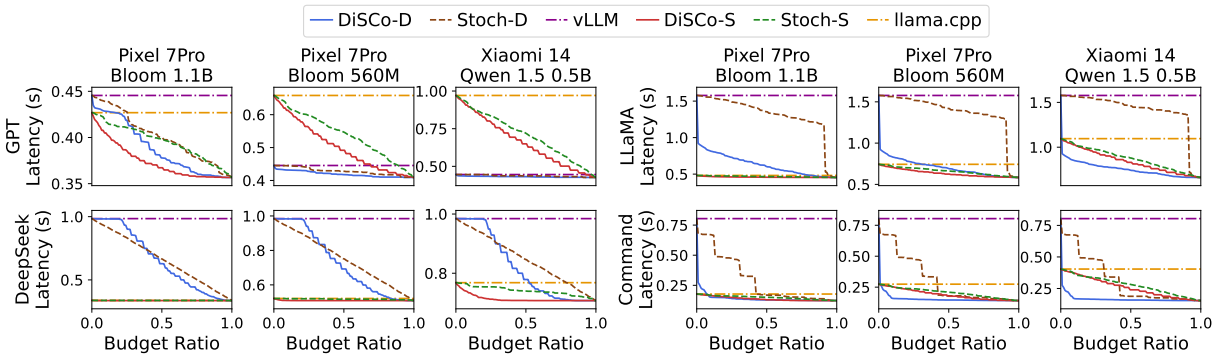


Figure 6: Mean TTFT tested using four traces. *DiSCo* achieves superior TTFT performance than the baselines.

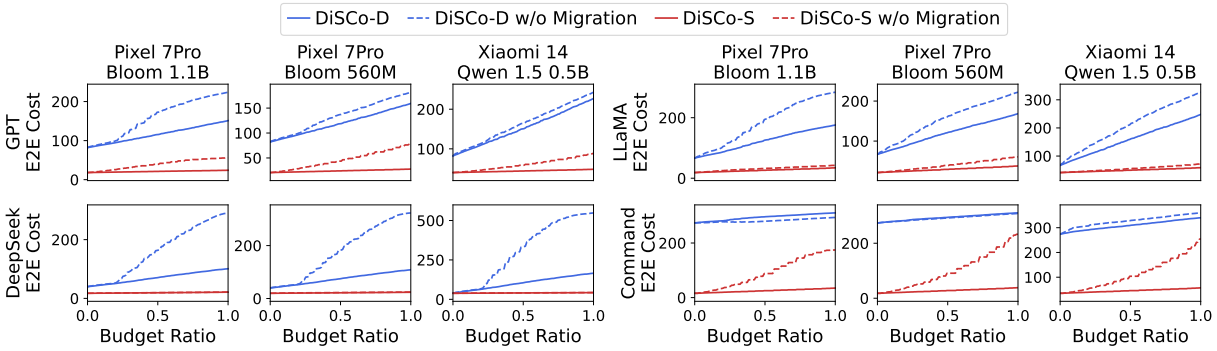


Figure 7: The migration mechanism in *DiSCo* achieves superior end-to-end cost.

Trace	Constraint	Mean delay_num	P99 delay_num	TBT P99
GPT	Server	4.21	9.40	0.209
	Device	6.59	6.59	0.217
LLaMA	Server	5.53	11.00	0.209
	Device	10.01	10.01	0.217
DeepSeek	Server	8.13	11.00	0.209
	Device	17.17	17.17	0.217
Command	Server	3.25	8.00	0.209
	Device	8.54	8.54	0.217

Table 3: Performance metrics for different models under server and device constraints, showing the number of delayed tokens during migration and TBT (Time Between Tokens) P99 statistics. The average is computed over the requests that have performed the migration.

latency. Results show that while migrations delay only a negligible number of tokens compared to typical generation lengths of hundreds or thousands of tokens, they do not impact the perceived token delivery smoothness, demonstrating *DiSCo*'s ability to maintain consistent streaming performance even during endpoint transitions.

As shown in Figure 7, our token-level migration mechanism substantially reduces the end-to-end cost across all evaluated scenarios. For device-constrained cases (*DiSCo*-D), the migration mechanism achieves up to 72.7% cost reduction compared to the non-migration baseline, with the improvement being most significant at higher budget ratios. Similarly, in server-constrained scenarios (*DiSCo*-S), the cost reduction reaches 83.6%, particularly evident in DeepSeek and Command model deployments. These significant cost reductions are consistently observed across device-model pairs.

5.3 Performance Breakdown and Ablation Study

Impact of Prompt Arrival Interval. To evaluate our system under realistic workload patterns, we conduct experiments using stratified sampling based on request arrival rate from DiffusionDB (Wang et al., 2022). Specifically, we select traces from ten users across different activity levels to capture diverse interaction patterns. We pair these real-world request intervals with prompts randomly drawn from the Alpaca dataset (Taori et al., 2023). The results shown in Figure 5 demonstrate that *DiSCo*'s performance advantages persist across varied user activity patterns.

Quality of Generated Responses. We conduct comprehensive experiments using instruction-following tasks on multiple model configurations. We employ three LLM-based judges (GPT-4o, Gemini1.5-pro, and QWen2.5-72b) to assess the response quality, and examine two representative migration scenarios: from a smaller to larger model (3B-7B) and vice versa (7B-3B). Figure 8 shows

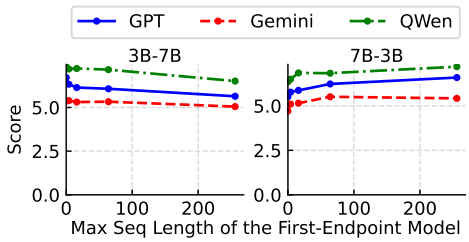
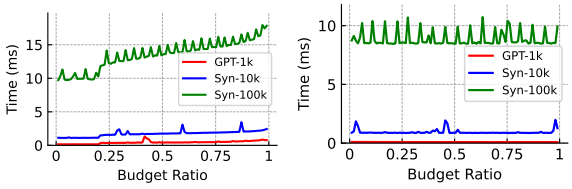


Figure 8: Response quality evaluation. Each subplot represents a distinct model pair configuration (e.g., 3B-7B indicates migration from a 3B to a 7B model). The x-axis shows the maximum sequence length processed by the first endpoint before migration, while the y-axis shows the quality scores assigned by different LLM judges. Results demonstrate consistent quality preservation across various migration scenarios.



(a) *DiSCo*-D (b) *DiSCo*-S

Figure 9: *DiSCo*'s overhead is trivial and can scale well.

that *DiSCo* maintains quality scores across different sequence lengths, migration patterns, and judges. Detailed results are presented in Appendix D.

Scalability Analysis. We conducted comprehensive performance evaluations of *DiSCo*-D and *DiSCo*-S on a MacBook Pro with M1 processor, using both synthetic datasets and a real-world GPT trace of 1,000 records, across target frequencies from 0 to 1. To generate synthetic data that accurately reflects real-world scenarios, we fitted log-normal distributions to the prompt lengths and TTFT from the real trace by following the mean and standard deviation of the logarithm. As shown in Figure 9, for *DiSCo*-S, the execution time showed remarkable efficiency: 0.128 ms for the real trace with 1K samples, scaling to just 0.969 ms and 9.082 ms for synthetic datasets of 10K and 100K samples, respectively. *DiSCo*-D, while being more computationally intensive, still maintained practical performance levels: 0.486 ms, 1.741 ms, and 14.856 ms for 1K, 10K, and 100K samples, respectively.

6 Conclusions

This paper introduces *DiSCo*, a device-server cooperative scheduler that addresses QoE and cost challenges in LLM serving for real-time conversational applications for end users. *DiSCo* uses cost-aware scheduling and token-level migration to dynamically optimize TTFT and TBT across device and server endpoints. Our evaluations on real-world traces from platforms like GPT and DeepSeek show that *DiSCo* significantly improves both TTFT and TBT while reducing costs.

7 Limitations

While *DiSCo* demonstrates significant improvements in LLM serving efficiency, we acknowledge several important limitations of our current work:

Model Accuracy. We focus on scenarios where on-device LLMs achieve sufficient accuracy for target applications. While this covers many common use cases, *DiSCo* may not be suitable for complex reasoning tasks (Guo et al., 2025).

Privacy Protection. While privacy protection is a key consideration in model selection (Chen et al., 2024; Liu et al., 2024b), *DiSCo* currently assumes users are comfortable with both on-device and on-server deployments.

Energy Modeling. For device energy consumption, we use a linear energy model based on FLOPs. Real-world device energy consumption patterns can be more complicated, varying on factors such as battery state, temperature, and concurrent workloads (Hoque et al., 2015).

Scalability. Our current implementation and evaluation focus on single-device scenarios. Extending *DiSCo* to handle multi-device collaborative serving presents additional challenges in terms of coordination overhead and resource allocation that warrant further investigation (Niu et al., 2025).

8 Ethical Considerations

Our work focuses solely on optimizing the efficiency of LLM serving systems through device-server collaboration and does not introduce new language generation capabilities or content. All experiments were conducted using publicly available models and datasets. While our work may indirectly benefit the accessibility of LLM services by reducing costs and improving performance, we acknowledge that broader ethical considerations around LLM deployment and usage are important but outside the scope of this technical contribution.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *Proceedings of 18th USENIX Symposium on Operating Systems Design and Implementation, 2024, Santa Clara*.

Alibaba. 2024. Qwen2.5 family. Accessed on 21 Sep 2024.

Artificial Analysis. 2025. Artificial analysis: Ai model performance benchmarking. https://artificialanalysis.ai. Accessed: 2025-05-17.

Barnard, Dom. 2022. Average speaking rate and words per minute. Accessed on 21 Sep 2024.

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

Daihang Chen, Yonghui Liu, Mingyi Zhou, Yanjie Zhao, Haoyu Wang, Shuai Wang, Xiao Chen, Tegawendé F Bissyandé, Jacques Klein, and Li Li. 2024. Llm for mobile: An initial roadmap. *ACM Transactions on Software Engineering and Methodology*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*.

Cohere. 2024. Command model. Accessed on 21 Sep 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Databricks. 2023. LLM Inference Performance Engineering: Best Practices. https://www.databricks.com/blog/\llm-inference-performance-engineering-best\-practices. [Online; accessed 27-September-2024].

DeepSeek. 2024. Deepseek-v2.5 model. Accessed on 21 Sep 2024.

DeepSeek. 2025. Deepseek: Open-source large language models. https://www.deepseek.com. Accessed: 2025-05-17.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Ali Diba, Vivek Sharma, Ali Pazandeh, Hamed Pirsiavash, and Luc Van Gool. 2017. Weakly supervised cascaded convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 914–922.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Georgi Gerganov. 2024. llama.cpp. Accessed on 21 Sep 2024.

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

Google. 2024. Gemini nano. Accessed on 21 Sep 2024.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Grand View Research. 2023. Large language model (llm) market size, share & trends analysis report by component, by application, by enterprise size, by end-use, by region, and segment forecasts, 2023 - 2030. Grand View Research.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet,

Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin,

Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, et al. 2024. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Ashit Gupta, Anirudh Deodhar, Tathagata Mukherjee, and Venkataramana Runkana. 2022. Semi-supervised cascaded clustering for classification of noisy label data. *arXiv preprint arXiv:2205.02209*.

Mohammad Ashraful Hoque, Matti Siekkinen, Kashif Nizam Khan, Yu Xiao, and Sasu Tarkoma. 2015. Modeling, profiling, and debugging the energy consumption of mobile devices. *ACM Computing Surveys (CSUR)*, 48(3):1–40.

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

Hyperbolic. 2024. Llama3-70b-instruct by hyperbolic. Accessed on 21 Sep 2024.

Hyperbolic. 2025. Hyperbolic inference service: Host custom ai models and open-source llms. `https://app.hyperbolic.xyz`. Accessed: 2025-05-17.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. 2024. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. *arXiv preprint arXiv:2402.07033*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Hanchen Li, Yuhan Liu, Yihua Cheng, Siddhant Ray, Kuntai Du, and Junchen Jiang. 2024a. Eloquent: A more robust transmission scheme for llm token streaming. In *NAIC*.

Xiang Li, Zhenyan Lu, Dongqi Cai, Xiao Ma, and Mengwei Xu. 2024b. Large language models on mobile devices: Measurements, analysis, and insights. In *Proceedings of the Workshop on Edge and Mobile Foundation Models*, EdgeFM '24.

Bin Lin, Tao Peng, Chen Zhang, Minmin Sun, Lanbo Li, Hanyu Zhao, Wencong Xiao, Qi Xu, Xiafei Qiu, Shen Li, et al. 2024a. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache. *arXiv preprint arXiv:2401.02669*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024b. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*.

Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. 2024a. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2404.16283*.

Xiaoze Liu, Ting Sun, Tianyang Xu, Feijie Wu, Cunxiang Wang, Xiaoqian Wang, and Jing Gao. 2024b. Shield: Evaluation and defense strategies for copyright compliance in llm text generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1640–1670.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. 2024c. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. 2024d. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2024e. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36.

Teemu Mäki-Patola and Perttu Hämäläinen. 2004. Latency tolerance for gesture controlled continuous sound instrument without tactile feedback. In *International Computer Music Conference*. Citeseer.

Microsoft Azure. 2025. Azure ai foundry: Model catalog and ai services. https://azure.microsoft.com/en-us/solutions/ai-foundry. Accessed: 2025-05-17.

MLC team. 2023. MLC-LLM.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Chaoyue Niu, Yucheng Ding, Junhui Lu, Zhengxiang Huang, Hang Zeng, Yutong Dai, Xuezhen Tu, Chengfei Lv, Fan Wu, and Guihai Chen. 2025. Collaborative learning of on-device small model and cloud-based large model: Advances and future directions. *arXiv preprint arXiv:2504.15300*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data.

OpenAI. 2022. tiktoken. Accessed on 21 Sep 2024.

OpenAI. 2024. Gpt-4o mini model. Accessed on 21 Sep 2024.

Tara Parachuk. 2022. Speaking rates comparison table. Accessed on 21 Sep 2024.

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

Aleksandar Petrov, Emanuele La Malfa, Philip H.S. Torr, and Adel Bibi. 2024. Language model tokenizers introduce unfairness between languages. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: Kimi's kvcache-centric architecture for llm serving. *arXiv preprint arXiv:2407.00079*.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. Powerinfer: Fast large language model serving with a consumer-grade gpu. *arXiv preprint arXiv:2312.12456*.

Ting Sun, Penghan Wang, and Fan Lai. 2025. Hygen: Efficient llm serving via elastic online-offline request co-location.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

NLLB Team, Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, et al. 2022. No language left behind: Scaling human-centered machine translation (2022). *URL https://arxiv.org/abs/2207.04672*.

Together AI. 2025. Together ai: High-performance inference for open-source llms. https://www.together.ai. Accessed: 2025-05-17.

Jakub Žádník, Markku Mäkitalo, Jarno Vanne, and Pekka Jääskeläinen. 2022. Image and video coding techniques for ultra-low latency. *ACM Comput. Surv.*

Zijie J. Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. 2022. DiffusionDB: A large-scale prompt gallery dataset for text-to-image generative models. *arXiv:2210.14896 [cs]*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. 2023. Llmcad: Fast and scalable on-device large language model inference. *arXiv preprint arXiv:2309.04255*.

Nan Xue, Yaping Sun, Zhiyong Chen, Meixia Tao, Xiaodong Xu, Liang Qian, Shuguang Cui, and Ping Zhang. 2024a. Wdmoe: Wireless distributed large language models with mixture of experts. *arXiv preprint arXiv:2405.03131*.

Zhenliang Xue, Yixin Song, Zeyu Mi, Le Chen, Yubin Xia, and Haibo Chen. 2024b. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*.

Zheming Yang, Yuanhao Yang, Chang Zhao, Qi Guo, Wenkai He, and Wen Ji. 2024. Perllm: Personalized inference scheduling with edge-cloud collaboration for diverse llm services. *arXiv preprint arXiv:2405.14636*.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*.

Mingjin Zhang, Jiannong Cao, Xiaoming Shen, and Zeyang Cui. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *arXiv preprint arXiv:2405.14371*.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*.

A Additional Related Work

General LLM Inference. LLMs generate text responses auto-regressively, producing one token at a time based on preceding tokens. The process consists of two stages that can potentially be executed on different endpoints: (i) *Prefill stage*: The model processes the input text (prompt), calculates and stores intermediate model states–i.e., the key and value cache (KV cache) of tokens–to generate the first token. A token represents a word or part of a word that the model can interpret. Once the first token is generated, it is appended to the end of the prompt and the generation process moves on to the (ii) *Decode stage*: The model processes the updated prompt (including previously generated tokens) to generate the next token. The decode stage continues until a stopping condition is met (e.g., reaching an end-of-sequence token or the maximum generation length).

On-Server LLM Serving. Existing works have focused on GPU kernel optimization (Dao et al., 2022; Ye et al., 2025), KV-cache management (Lin et al., 2024a; Liu et al., 2024e; Qin et al., 2024), model parallelism (Shoeybi et al., 2019; Pope et al., 2023; Liu et al., 2023), quantization (Xiao et al., 2023; Lin et al., 2024b; Dettmers et al., 2022), and scheduling (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024; Sun et al., 2025). For example, Orca (Yu et al., 2022) introduced continuous batching to improve serving throughput, while vLLM (Kwon et al., 2023) developed PagedAttention to reduce LLM memory restraint. Sarathi (Agrawal et al., 2024) implemented chunked prefill to mitigate inter-request interference within batches. Andes (Liu et al., 2024a) addresses QoE for individual requests from the server side but lacks awareness of network jitter and device potential. These server-side advancements complement *DiSCo*'s design.

On-Device LLMs. Google's Gemini Nano (Google, 2024) and Apple's Apple Intelligence (Gunter et al., 2024) have been integrated into Android OS and iOS devices, respectively. MLC-LLM (MLC team, 2023) and llama.cpp (Gerganov, 2024) efficiently deploy various LLMs on devices. PowerInfer (Song et al., 2023) and PowerInfer-2 (Xue et al., 2024b) optimize on-device LLM inference by leveraging sparsity in model activations. *DiSCo* acts as a middle layer to schedule and migrate response generation between servers and devices.

B Cold Start Evaluation

This section presents cold start performance measurements for the Qwen-2.5 model series across different hardware configurations. The experiments were conducted on two platforms: Windows 10 with NVIDIA RTX 3060 12GB and Linux with NVIDIA A40 48GB. A fixed prompt "How to use GitHub?" was used throughout all experiments. We measured two critical metrics: model loading time and TTFT for Qwen-2.5 models ranging from 0.5B to 7B parameters, all using FP16 precision. The experimental setup consisted of 10 measurement runs, with two additional warmup runs to ensure measurement stability. It is worth noting that such warmups can potentially mask the true gap between model loading and prompt prefill time due to var-

ious optimizations, including OS page cache. To maintain authentic cold start conditions, we explicitly cleared the CUDA cache and performed garbage collection before each run.

The results revealed several significant patterns. On the RTX 3060, the loading time exhibits an approximately linear increase with model size, ranging from 1.29s for the 0.5B model to 4.45s for the 3B model. While TTFT follows a similar trend, the processing time is substantially lower, ranging from 0.051s to 0.145s. On the A40 GPU, despite observing longer loading times, TTFT is significantly reduced across all models, maintaining a remarkably consistent value regardless of model size. These findings indicate that while model loading remains more resource-intensive on our Linux setup, the inference performance benefits substantially from the A40's superior computational capabilities.

Metric	0.5B	1.5B	3B	7B
Windows 10 (NVIDIA RTX 3060 12GB)				
Load Time (s)	1.29	2.48	4.45	-
TTFT (s)	0.051	0.105	0.145	-
Linux (NVIDIA A40 48GB)				
Load Time (s)	1.53	3.12	5.72	13.43
TTFT (s)	0.025	0.026	0.033	0.033

Table 4: Model loading time during cold start can significantly slow down TTFT. Average Qwen-2.5 model performance over 10 runs. The 7B model exceeds the memory capacity of the RTX 3060 and thus cannot be evaluated.

C Prediction-based Model Selection

This section provides a comparative analysis of several TTFT prediction methods. For selecting the endpoint with a lower TTFT for each request, TTFT prediction is imperative. For on-device inference, TTFT prediction is straightforward, as TTFT exhibits a linear relationship with prompt length. Conversely, on-server TTFT is characterized by high variability, posing challenges for prediction. Moreover, the prediction method itself must be computationally efficient, as its overhead also contributes to end-to-end TTFT.

Table 5 presents a comparative analysis of four common lightweight time-series-based prediction methods applied to traces collected from three prevalent LLM services. Our correlation analysis (Table 1) reveals no significant correlation between prompt length and TTFT; thus, prompt length is omitted as a feature in these prediction methods. We demonstrate that none of these methods offers sufficient accuracy for TTFT prediction.

Model	MAPE(%)	MAE(s)
Command		
Moving Average	39.40	0.0899
ExponentialSmoothing	53.51	0.1047
Random Forest	39.33	0.0966
XGBoost	35.43	0.0905
DeepSeek-V2.5		
Moving Average	27.80	0.3959
ExponentialSmoothing	27.39	0.3771
Random Forest	32.97	0.4745
XGBoost	27.51	0.4001
GPT-4o-mini		
Moving Average	24.55	0.0995
ExponentialSmoothing	20.88	0.0844
Random Forest	28.68	0.1128
XGBoost	24.83	0.0997
LLaMA-3-70b-Instruct		
Moving Average	42.18	0.3312
ExponentialSmoothing	40.27	0.3154
Random Forest	49.67	0.3875
XGBoost	43.94	0.3451

Table 5: Comparative analysis of Moving Average, Exponential Smoothing, Random Forest, and XGBoost prediction models across Command, DeepSeek, GPT, and LLaMA model traces. Metrics include Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE).

To address potential concerns about the robustness of our server-side TTFT prediction under varying workload conditions, we emphasize that our distribution-based prediction model is specifically designed to handle workload bursts and dynamic patterns. The statistical distribution we maintain inherently captures these patterns, allowing the model to automatically adapt to changing workloads based on historical data. Furthermore, our system architecture supports dynamic distribution updates through a lightweight server API, which can efficiently transmit real-time server workload information using just tens or hundreds of numbers. This design ensures that our prediction model remains responsive and accurate even during significant load spikes or ephemeral workloads.

D Response Quality

This section examines the quality of responses generated by *DiSCo*, with a particular focus on quality preservation during endpoint transitions. We first

establish bounds on generation quality, then present our evaluation methodology, and finally demonstrate through extensive experiments that *DiSCo* maintains consistent quality across different model configurations and tasks.

D.1 Quality Bounds

A critical aspect of *DiSCo* is maintaining generation quality during endpoint transitions. We employ a systematic approach to quality preservation (Diba et al., 2017; Gupta et al., 2022; Chen et al., 2023). Specifically, for endpoints A and B with quality metrics Q_A and Q_B (measured by LLM scores or ROUGE scores), we find that any migrated sequence M with quality Q_M satisfies:

$$\min(Q_A, Q_B) \leq Q_M \leq \max(Q_A, Q_B) \quad (7)$$

This bound ensures that migration does not degrade quality beyond the capabilities of individual endpoints.

D.2 Evaluation Methodology

Evaluation Framework We establish a comprehensive assessment framework encompassing both automated metrics and LLM-based evaluation. Our framework evaluates two distinct tasks:

- **Translation Quality**: We assess Chinese-to-English translation on 500 data items from Flores_zho_Hans-eng_Latn dataset (Team et al., 2022; Goyal et al., 2022) using the ROUGE-1 metric.
- **Instruction Following**: We evaluate 500 data items from the Alpaca dataset (Taori et al., 2023) using our structured prompt template, with quality assessment performed by multiple LLM judges: Gemini1.5-pro, GPT-4o, and QWen2.5-72b-instruct.
- **Text Summarization**: We evaluate 100 data items from the CNN/DM (Nallapati et al., 2016) using our structured prompt template, with quality assessment performed by DeepSeek-V3-0324.
- **Story Writing**: We evaluate 100 data items from the WritingPrompts (Fan et al., 2018) using our structured prompt template, with quality assessment performed by DeepSeek-V3-0324.

These two tasks are popular on end-user devices. Understandably, for complex tasks such as advanced math reasoning, we notice DisCo can lead to accuracy drops compared to the on-server model due to the limited capability of the on-device models, yet still achieves better performance than the on-device counterpart.

Experimental Setup We configure our experiments with:

- A fixed maximum generation length of 256 tokens
- First endpoint's maximum generation length varied through different (e.g., [0, 4, 16, 64, 256] tokens).
- Four model combinations: 0.5B-7B, 3B-7B, 7B-0.5B, and 7B-3B (prefix and suffix denote the model sizes of first and second endpoints, respectively)

The generation transitions to the second endpoint when the first endpoint reaches its length limit without producing an end-of-generation token, creating natural boundary conditions for analysis.

For instruction-following tasks, we employ the following structured evaluation template:

```
JUDGE_PROMPT = """Strictly evaluate the
quality of the following answer on a scale
of 1-10 (1 being the worst, 10 being the
best). First briefly point out the problem
of the answer, then give a total rating in
the following format.

Question: {question}

Answer: {answer}

Evaluation: (your rationale for the rating,
as a brief text)

Total rating: (your rating, as a number
between 1 and 10)
"""
```

D.3 Results and Analysis

D.3.1 Quality Metrics

Our comprehensive evaluation reveals that the combined sequences' quality consistently remains bounded between individual model performance levels.

E Experiment Settings for End-to-end Cost

For on-device LLMs, we quantify cost using FLOPs (floating-point operations). For on-server

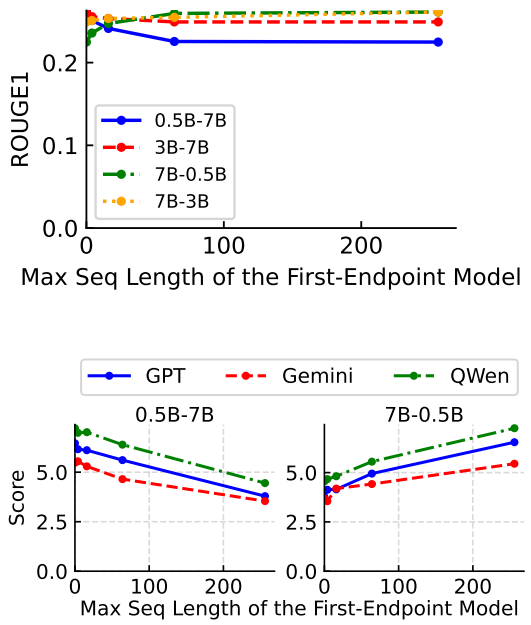
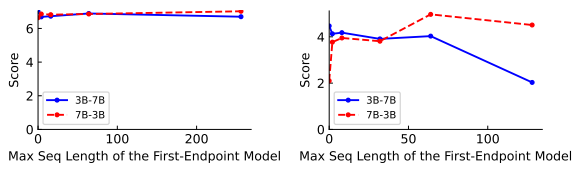


Figure 10: Quality evaluation results of *DiSCo*. The top figure shows translation quality evaluation using ROUGE-1 scores, demonstrating that *DiSCo* consistently achieves higher quality than the on-device baseline. The bottom figure presents evaluation scores from different LLM judges on instruction-following capabilities, where each subplot represents a different model pair comparison with varied first-endpoint model's maximum sequence length. The consistent patterns across different LLM judges demonstrate the robustness of our evaluation framework.

LLM services, we use their respective pricing rates at the time of experimentation. We set the energy-to-monetary conversion ratio (*energy_to_money*) to 0.3 $ per million FLOPs for server-constrained experiments and 5 $ per million FLOPs for device-constrained experiments. To establish a comprehensive cost model that enables direct comparison between device and server computation costs, we analyze both the computational complexity of on-device models through detailed FLOPs calculations (Section E.2) and the pricing structures of commercial LLM services (Section E.3). The generation length limit is set to 128.



(a) Summarization. (b) Story Writing.

Figure 11: Quality evaluation results with Summarization and Story Writing.

E.1 Theoretical Cost Modeling

Our unified cost model combines heterogeneous costs (server monetary and device energy) through a dynamic exchange rate λ, which users adjust based on preferences (e.g., battery level, server budget). We formalize this as:

$$\text{Total Cost} = \underbrace{c_s^p \cdot l_s + c_s^d \cdot l_s}_{\text{Server Cost}} + \lambda \cdot \underbrace{\left(c_d^p \cdot l_d + c_d^d \cdot l_d\right)}_{\text{Device Cost}} \tag{8}$$

where l_s, l_d denote tokens processed on the server and the device, respectively. The optimization goal is to minimize $\mathbb{E}[\text{Total Cost}]$ while satisfying TTFT/TBT constraints. This formulation allows us to:

- Balance between server monetary costs and device energy consumption
- Adapt to user preferences through the exchange rate λ
- Account for different costs in prefill (c^p) and decode (c^d) phases
- Optimize token allocation between server and device execution

The model's flexibility enables it to handle both device-constrained scenarios (where energy consumption is the primary bottleneck) and server-constrained scenarios (where API monetary costs dominate).

While we acknowledge that real-world device energy consumption can be influenced by factors such as battery levels, thermal throttling, and concurrency, we use the linear FLOPs-based measure as a generalizable proxy metric for two key reasons: (1) to cover energy consumption across diverse devices (phones, laptops, and edge servers) and models with varying architectures and quantizations that may exhibit different consumption patterns, and (2) to target short conversational tasks where on-device LLMs demonstrate sufficient efficiency and effectiveness. Importantly, since our dispatching and migration algorithms are both length-threshold-based, they can adapt to any complex energy consumption patterns as long as these patterns are predictable and guaranteed to consume more energy when prefilling or decoding additional tokens.

E.2 FLOPs of On-Device LLMs

To accurately quantify the computational cost per token in both prefill and decode stages, we conduct a detailed FLOPs analysis using three representative models: BLOOM-1.1B, BLOOM-560M, and Qwen1.5-0.5B. All models share a 24-layer architecture but differ in other parameters: BLOOM-1.1B ($d_{\text{model}} = 1024$, 16 heads, FFN dim=4096), BLOOM-560M ($d_{\text{model}} = 512$, 8 heads, FFN dim=2048), and Qwen1.5-0.5B ($d_{\text{model}} = 768$, 12 heads, FFN dim=2048).

Per-token FLOPs computation. The total FLOPs for processing each token consist of five components:

$$\begin{aligned}\text{FLOPs}_{\text{total}} = \text{FLOPs}_{\text{attn}} + \text{FLOPs}_{\text{ffn}} \\ + \text{FLOPs}_{\text{ln}} + \text{FLOPs}_{\text{emb}} + \text{FLOPs}_{\text{out}}\end{aligned} \tag{9}$$

For a sequence of length L, the attention computation differs between stages. In prefill:

$$\begin{aligned}\text{FLOPs}_{\text{attn}} = n_{\text{layers}} \cdot \Big(3d_{\text{model}}^2 + \frac{L^2 d_{\text{model}}}{n_{\text{heads}}} \\ + Ld_{\text{model}} + d_{\text{model}}^2\Big)\end{aligned} \tag{10}$$

While in decode, KV caching eliminates the quadratic term:

$$\begin{aligned}\text{FLOPs}_{\text{attn}} = n_{\text{layers}} \cdot \Big(3d_{\text{model}}^2 + \frac{Ld_{\text{model}}}{n_{\text{heads}}} \\ + Ld_{\text{model}} + d_{\text{model}}^2\Big)\end{aligned} \tag{11}$$

Table 6 presents the total FLOPs across different sequence lengths. The decode phase maintains constant FLOPs regardless of sequence length due to KV caching, while prefill phase FLOPs increase with sequence length. A breakdown of computational cost by component (Table 7) reveals that embedding and output projection operations account for the majority of FLOPs, particularly in models with large vocabularies.

E.3 LLM Service Pricing

This section provides further details on the pricing of LLM services. Table 8 presents the pricing models for several commercial Large Language Models (LLMs) as of October 28, 2024. The pricing structure follows a dual-rate model, differentiating between input (prompt) and output (generation) tokens. These rates represent the public pricing tiers available to general users, excluding any enterprise-specific arrangements or volume-based discounts.

Length	BLOOM-1.1B	BLOOM-560M	Qwen-0.5B
Prefill Phase			
L = 32	0.85	0.45	0.39
L = 64	0.93	0.50	0.45
L = 128	1.25	0.65	0.69
Decode Phase			
L = 32	0.82	0.42	0.37
L = 64	0.82	0.42	0.37
L = 128	0.82	0.42	0.37

Table 6: Prefill and Decode FLOPs (billions)

Component	BLOOM-1.1B	BLOOM-560M	Qwen-0.5B
Embedding	31.24	25.00	31.51
Attention	13.01	10.00	16.56
FFN	24.48	20.00	20.38
LayerNorm	0.02	0.02	0.04
Output	31.24	25.00	31.51

Table 7: Component Ratios at L=128 (%)

F Pseudocode for Cost-Aware Adaptive Request Scheduling

The request scheduling algorithm consists of three key components. Algorithm 1 defines the input parameters and determines whether the scenario is device-constrained or server-constrained based on the relative costs. For device-constrained scenarios, Algorithm 2 implements a wait-time strategy to protect tail latency while conserving device energy when possible. For server-constrained scenarios, Algorithm 3 employs a length-based routing approach to optimize TTFT while maintaining the server budget constraint. These algorithms work together to achieve the dual objectives of minimizing latency and managing costs.

Algorithm 1 Variable Definitions and Constraints

Require:

$p(l)$: Length distribution
$F(t)$: TTFT CDF of server
$b \in [0, 1]$: Budget ratio
c_d^p, c_d^d: Device prefill/decode costs
c_s^p, c_s^d: Server prefill/decode costs
$\alpha \in (0, 1)$: Tail ratio

Ensure: Policy type based on cost constraints

if $\min(c_d^p, c_d^d) > \max(c_s^p, c_s^d)$ **then** Device-constrained
else Server-constrained

Model	Vendor	Input price	Output price
DeepSeek-V2.5	DeepSeek	0.14	0.28
GPT-4o-mini	OpenAI	0.15	0.60
LLaMa-3.1-70b	Hyperbolic	0.40	0.40
LLaMa-3.1-70b	Amazon	0.99	0.99
Command	Cohere	1.25	2.00
GPT-4o	OpenAI	2.50	10.0
Claude-3.5-Sonnet	Anthropic	3.00	15.0
o1-preview	OpenAI	15.0	60.0

Table 8: LLM service pricing (USD per 1M Tokens). Input prices refer to tokens in the prompt, while output prices apply to generated tokens.

Algorithm 2 Device-constrained Scheduling

Require: Variables from Algorithm 1

// Phase 1: Set maximum wait time for tail protection
$w_{tail} \leftarrow F^{-1}(1 - \min(\alpha, b))$
// Initialize wait times for all prompt lengths
$W \leftarrow \{l : w_{tail}$ for all $l\}$
if $b \leq \alpha$ **then**
  return W {Use max wait time for all lengths}
end if
// Phase 2: Optimize wait times with remaining budget
available_budget $\leftarrow b - \alpha$
for $l \in$ sort(support($p(l)$)) **do**
  length_cost $\leftarrow p(l) \cdot l \cdot (1 - \alpha)$
  if available_budget $\geq$ length_cost **then**
    $W[l] \leftarrow 0$ {Start device immediately}
    available_budget $\leftarrow$ available_budget - length_cost
  else
    // Find optimal wait time that meets budget
    Find $w^* \in [0, w_{tail}]$ where:
    $F(w^*)\cdot$ length_cost + (b - available_budget) = b
    $W[l] \leftarrow w^*$
    break
  end if
end for
return W {Map from prompt lengths to wait times}

Algorithm 3 Server-constrained Scheduling

Require: Variables from Algorithm 1

// Find length threshold to split execution modes
Compute l_{th} where: $\int_0^{l_{th}} l \cdot p(l)dl = (1 - b) \cdot \int_0^{\infty} l \cdot p(l)dl$
// Initialize execution policy map
$P \leftarrow \emptyset$
for $l \in$ support($p(l)$) **do**
  if $l < l_{th}$ **then**
    $P[l] \leftarrow$ (1, 0) {(I_d, I_s): Device only}
  else
    $P[l] \leftarrow$ (1, 1) {(I_d, I_s): Concurrent execution}
  end if
end for
return P {Map from lengths to execution indicators}