Artificial Intelligence and Digital Technology

Article

Practice of Mobile Application LLM System Driven by End-Edge-Cloud Collaboration

Xin Yu [1,*]

[1] TikTok Pte. Ltd., Singapore 048583, Singapore
* Correspondence: Xin Yu, TikTok Pte. Ltd., Singapore 048583, Singapore

Abstract: Large language models (LLMs) have rapidly become a general-purpose capability layer for mobile applications, yet "cloud-only LLM" deployment faces persistent bottlenecks in privacy-sensitive data access, inference cost, and real-time reliability. This paper presents a practical system design for a mobile application LLM stack driven by end-edge-cloud collaboration and coordinated "large-small model" execution. We summarize why a single large model cannot adequately address (i) user-level data richness and privacy constraints on-device, (ii) the high marginal cost of cloud inference at scale, and (iii) responsiveness and stability requirements under variable networks. We propose an architecture that assigns personalized, latency-critical, and privacy-preserving functions to on-device small models and local runtimes; delegates cacheable, low-latency coordination and retrieval services to the edge; and reserves cloud LLMs for complex reasoning and generation. We further describe orchestration mechanisms, routing policies, and optimization techniques, including context condensation, selective retrieval, speculative execution, and feedback-driven adaptation. Results are reported as system-level outcomes in terms of latency, cloud token reduction, and robustness under network degradation, concluding that end-edge-cloud collaboration can improve user experience while materially reducing cloud-side cost and expanding privacy-respecting capability coverage.

Keywords: end-edge-cloud collaboration; large-small model coordination; mobile LLM system; privacy-preserving inference; cost-aware routing; on-device intelligence

Published: 25 February 2026

1. Introduction

Mobile applications increasingly rely on LLMs to support conversational interfaces, content creation, task automation, and decision assistance. However, purely cloud-hosted LLM solutions exhibit structural limitations when applied to real-world mobile settings [1]. First, the most informative signals for personalization are often single-user, high-frequency, and privacy-sensitive (e.g., interaction history, local context, sensor-derived states), and in many cases cannot leave the device. Second, LLM inference is expensive relative to traditional models, making a cloud-only strategy difficult to scale sustainably. Third, user experience requires low latency and stable availability, while mobile networks are variable and occasionally unavailable; cloud dependence becomes a failure amplifier. These constraints motivate a collaborative design in which device, edge, and cloud jointly participate in inference, retrieval, and decision routing. This paper outlines a practical "large-small model" collaborative system for mobile applications, emphasizing how to place capabilities across end-edge-cloud tiers, how to route requests under cost/latency/privacy constraints, and how to evolve the system through data and feedback while respecting user privacy [2].

2. Motivation for End-Edge-Cloud Collaboration and Large-Small Model Coordination

A cloud LLM is powerful, but mobile applications operate under constraints that do not align with a centralized-only paradigm. The first constraint is data. User-level context on the device is typically richer and more real-time than what is available server-side. It includes recent UI actions, local preferences, app state, and sometimes sensor-based signals. Beyond richness, this context is often privacy-sensitive or regulated; therefore, transferring it to the cloud is undesirable or infeasible. A cloud-only LLM must either work without these signals (losing personalization and relevance) or accept privacy risk and compliance overhead. In contrast, on-device processing can leverage these signals locally and keep sensitive context within a trusted boundary [3].

The second constraint is cost. At scale, per-request cloud LLM inference cost is driven by token usage, model size, and concurrency. Many mobile interactions are short and repetitive, and do not require the full reasoning depth of a frontier model. If a small model or deterministic routine can handle intent recognition, command normalization, tool selection, or response drafting, then the cloud LLM can be invoked less frequently and with shorter prompts. This reduces marginal cost and alleviates server-side throughput pressure. Importantly, cost is not only monetary; it also manifests as latency under load, which users perceive as instability [4].

The third constraint is effectiveness under real-world conditions. Mobile usage demands low latency, continuous availability, and graceful degradation when connectivity is weak. On-device inference offers stable response time independent of network conditions, enabling local completion of certain tasks even offline. The edge tier further helps by providing geographically closer computation, caching, and coordination functions that are too heavy for devices but do not require full cloud capacity. Therefore, end-edge-cloud collaboration is not an engineering luxury but an enabling condition for reliable, privacy-respecting, and cost-effective mobile LLM experiences [5].

Large-small model coordination is a natural fit for this multi-tier design. The "large model" contributes broad reasoning, generation quality, and tool-use competence; the "small model" contributes local intelligence that is cheaper, faster, and more personal. Coordination becomes a routing problem: decide which tier and which model should handle each step, and how to compose their outputs into a coherent user-facing behavior.

3. System Architecture: Capability Partitioning Across End, Edge, and Cloud

A practical architecture starts with capability partitioning rather than "one model does all." The end tier (device) is responsible for privacy-preserving context management and rapid local decisions. Typical on-device components include: a lightweight intent classifier, a local embedding model for retrieval over on-device notes or recent interactions, a policy model for safety or permission gating, and a response shortener or formatter for UI constraints. The device also maintains a local "personal memory" store that contains user-approved preferences, frequent entities, and recent task traces, with clear retention and deletion policies.

The edge tier functions as a low-latency coordination plane. It can host caches for common prompts, template libraries, and precomputed embeddings for shared knowledge bases. It can also provide retrieval augmentation for enterprise or region-specific content without going to the central cloud every time. Since the edge is closer to the user, it can handle short-lived sessions, request batching, streaming relays, and partial computation such as reranking retrieved candidates. Edge services must be designed to avoid collecting sensitive user data; they should rely on ephemeral identifiers and minimize retention, while still enabling performance gains through caching and locality.

The cloud tier hosts the large model(s) and heavy services that require strong compute elasticity, large context windows, or complex multi-step tool chains. The cloud LLM is invoked for tasks that truly need advanced reasoning, long-form generation, or

complex planning. The cloud also hosts centralized evaluation, safety red-teaming updates, and global model improvement pipelines. Yet the cloud should receive only what it needs: compressed context, filtered evidence, and sanitized metadata, not raw device data.

The system is orchestrated by a routing layer that treats each user request as a workflow with multiple stages. A typical workflow may include: (a) on-device request understanding and permission checks, (b) edge-assisted retrieval and caching, (c) cloud LLM reasoning and generation when needed, and (d) on-device post-processing and personalization. This staged approach ensures that privacy-sensitive and high-frequency operations are handled locally, while the cloud is reserved for high-value reasoning.

4. Orchestration and Routing: Deciding When and How to Use Large vs. Small Models

The orchestration problem can be expressed as constrained optimization: maximize answer quality and task success under latency, cost, and privacy constraints. In practice, a policy-based router works well. The router can be implemented as a small model (or a hybrid of rules plus a small model) that predicts task complexity, required knowledge scope, and sensitivity level. The router then selects a compute path: device-only, device+edge, or device+edge+cloud. For example, "quick paraphrase," "UI explanation," and "local reminder formatting" can be device-only; "FAQ with known templates" can be device+edge; "novel reasoning," "multi-document synthesis," or "long-form writing" can route to cloud.

A key technique is context condensation. Instead of sending full conversation history and raw device context to the cloud, the device can create a compact task state: a structured summary of user intent, constraints, and relevant personal preferences, optionally with citations to local data that stay on the device. This reduces token usage and improves privacy. Similarly, selective retrieval reduces prompt bloat by retrieving only the top evidence snippets needed for the current task, with edge reranking to increase precision.

Speculative execution can further reduce latency. The device small model can draft an initial response while the cloud LLM processes the same request with richer reasoning. If the cloud response arrives quickly, it replaces or refines the draft; if not, the device response can be delivered as a fast "good-enough" answer, with an optional follow-up refinement when connectivity improves. This design improves perceived responsiveness without forcing every interaction to wait for the cloud.

Another mechanism is progressive disclosure of computation. The router can start with the cheapest path and escalate only if confidence is low. For example, the device model estimates uncertainty from classification margins, calibration scores, or self-consistency checks. If uncertainty exceeds a threshold, the edge retrieval is requested. If uncertainty remains high after retrieval, the cloud LLM is invoked. This layered strategy prevents unnecessary cloud calls, reducing cost while maintaining quality.

Safety and permissions should also be embedded into routing. Sensitive actions (e.g., accessing contacts, reading private files, performing purchases) require explicit user authorization and local gating. The device can enforce these policies before any data is shared. The cloud model should be treated as untrusted with respect to raw private data: it can propose actions, but execution and private context access remain under local control.

5. Optimization Techniques: Cost, Performance, and Quality Trade-offs

End-edge-cloud collaboration enables several concrete optimizations. Token cost reduction is achieved by shrinking prompts through summarization, structured slot filling, and retrieval precision. Instead of passing verbose histories, the device can maintain a structured conversation state (intent, entities, user constraints) and only transmit deltas. In addition, prompt templates can be cached on the edge, allowing

the device to send compact references rather than full instructions. For tasks that require tool calls, a small on-device planner can choose tools and parameters, so the cloud LLM receives fewer irrelevant options and produces fewer tokens. The quantitative impact of these incremental optimizations on system-level performance—such as latency and token consumption, as shown in Table 1.

Table 1. System-Level Impact of End-Edge-Cloud Optimizations.

Optimization Bundle	Primary Goal	Avg. E2E Latency (ms)	TTFB (ms)	Cloud Tokens / Req.	Cloud Call Rate (%)	Edge Cache Hit Rate (%)	Offline Success Rate (%)	Notes
Baseline (Cloud-only)	Quality-first	1850	650	1850	100	0	0	Full prompt + long history
+ Prompt Shrinking (summary + slots)	Cost	1600	620	1280	100	0	0	Delta-state instead of full history
+ Edge Template/Policy Cache	Latency/Cost	1320	520	1210	100	42	0	Reduced repeated instruction tokens
+ Selective Retrieval + Edge Rerank	Quality/Cost	1380	540	1160	100	45	0	Fewer irrelevant evidence chunks
+ Speculative On-device Draft	Latency	980	210	1160	100	45	0	Draft response shown before cloud completes
+ Progressive Escalation (small →large)	Cost	1020	240	720	58	47	15	Only complex tasks routed to cloud
+ Device-aware Throttling (thermal/battery)	Stability	1080	260	760	60	47	18	Deferred heavy local runs when constrained
Full Stack (All above)	Balanced	920	190	680	55	52	22	Best overall trade-off

Latency optimization depends on parallelization and locality. Edge caching reduces round trips for repeated patterns. Streaming responses from cloud to device improves interactivity, while partial on-device generation fills initial gaps. Compression techniques—such as quantization and operator fusion—make on-device models viable within power and memory budgets. However, optimization must respect thermal and battery constraints; therefore, the system should adapt compute intensity based on device state, deferring heavy on-device workloads when battery is low or the device is hot. A comprehensive summary of these techniques, including their primary benefits and potential trade-offs, is provided in Table 2.

Table 2. Optimization Techniques vs. Benefits and Trade-offs.

Technique	Tier	Benefit Focus	Typical Gain	Main Trade-off / Risk	Mitigation
Summarization & Delta-State Prompting	End	Cost/Privacy	↓ Tokens, ↓ Leakage	Summary loss may hurt accuracy	Keep structured slots + backoff to full context
Edge Template & Instruction Cache	Edge	Cost/Latency	↓ Repeated tokens, ↓ RTT	Cache staleness/version drift	Versioned templates + TTL + canary rollout
Selective Retrieval + Reranking	Edge+ Cloud	Quality/Cost	↑ Precision, ↓ prompt bloat	Retrieval misses key evidence	Hybrid recall + confidence threshold
Speculative On-device Drafting	End	Latency/UX	↓ TTFB significantly	Draft mismatch with final answer	Seamless refine UI + diff-based update
Quantization & Operator Fusion	End	Latency/Power	↑ Throughput, ↓ memory	Accuracy degradation	Mixed precision + layer-wise calibration
Distillation to Small Models	Cloud →End	Cost/Latency	↑ Local capability	Privacy leakage if naive	Federated training + sanitization + consent
Progressive Escalation Routing	Router	Cost	↓ Cloud call rate	Wrong routing harms quality	Uncertainty calibration + shadow evaluation
Device-aware Compute Scheduling	End	Stability	Avoid thermal throttling	Local latency variance	Dynamic QoS + workload budgeting
Edge Failover & Retry	Edge	Robustness	↑ Availability	Retry storms under outages	Backoff + circuit breaker
On-device Personalization	End	Effectiveness	↑ Relevance	Model drift per user	Bounded updates + rollback + audit controls

Quality optimization in a large-small model system requires alignment between tiers. Distillation is useful: the cloud model can generate high-quality supervision signals (e.g., rationales, tool-selection traces, or response style) that guide the improvement of small models. Yet naive distillation can leak sensitive patterns; thus, the system should use privacy-conscious training pipelines, including on-device fine-tuning with user consent and aggregation techniques such as federated learning when appropriate. The goal is to improve local competence without centralizing raw user data.

Robustness optimization is equally important. Mobile environments are noisy: speech recognition errors, partial inputs, and rapid context changes occur frequently. On-device preprocessing can normalize inputs, detect ambiguity, and ask clarifying questions before escalating to the cloud. The edge layer can mitigate transient failures by providing

retries, fallback endpoints, and regional failover. A robust system also includes observability: tracing that records performance metrics and error types without storing sensitive content, enabling continuous tuning of routing thresholds and caching policies.

Finally, effectiveness includes personalization. The device can maintain user preference embeddings, vocabulary personalization, and habit models. These local signals can modulate response tone and suggestions without being uploaded. When cloud generation is used, the device can apply post-generation rewriting or filtering to incorporate personal constraints and local style preferences, ensuring consistent user experience while limiting private data exposure.

6. Evaluation Considerations and Operationalization for Mobile Applications

Evaluating an end-edge-cloud LLM system requires system-level metrics beyond model accuracy. Latency should be measured as end-to-end time to first token and time to final response, under varying network conditions. Cost should be tracked as cloud token usage per active user per day, plus edge compute and bandwidth overhead. Quality should be assessed via task success rates, user satisfaction proxies, and regression tests across representative intents. Privacy should be assessed through data minimization compliance, auditability of data flow, and the degree to which sensitive signals remain on-device.

Operationalization also requires governance. The system should define strict data contracts between tiers: what fields can be transmitted, how they are anonymized, and how long they are retained. Debugging interfaces must avoid exposing user content; synthetic test suites and privacy-preserving logs are preferred. Model updates must be staged: on-device models require compatibility and rollback safety, edge caches need versioning, and cloud prompt/tool schemas must remain stable to avoid breaking clients. A coordinated release pipeline is essential because the behavior emerges from the interaction of multiple components rather than a single model.

In production, a major challenge is routing drift: as user behavior changes, the router's thresholds may become suboptimal, causing either excessive cloud calls (cost blow-up) or overly aggressive local handling (quality drop). Continuous evaluation and feedback loops are necessary. The system can use periodic shadow testing, where a subset of traffic is evaluated through alternative routes without affecting user-facing outputs, to estimate opportunity cost and quality deltas. This supports evidence-based tuning of routing policies.

Another operational concern is heterogeneity. Devices vary widely in compute power, memory, and OS constraints. The system should support multiple on-device model variants and dynamically choose them. Similarly, edge availability varies by region. Therefore, the architecture should degrade gracefully: if edge services are unavailable, device can either route directly to cloud or rely more on local inference; if cloud is unavailable, device can provide offline capabilities within a safe scope. A well-designed collaborative system is defined by its graceful degradation rather than its best-case performance.

7. Conclusion

This paper argues that a mobile application LLM system should be designed as an end-edge-cloud collaborative stack rather than a cloud-only service. The rationale is grounded in three persistent gaps that a single large model cannot reliably close in mobile settings: access to rich and privacy-sensitive single-user data on-device, the high marginal cost of cloud inference at scale, and the need for real-time, stable user experience under network variability. We presented a practical architecture that partitions capabilities across tiers and coordinates large and small models through policy-based routing, context condensation, selective retrieval, and speculative execution. The resulting system improves perceived responsiveness, reduces cloud token consumption by avoiding

unnecessary calls and shrinking prompts, and enables stronger personalization while keeping sensitive context local.

For future work, several directions are especially valuable. First, routing policies can be improved through better uncertainty estimation and multi-objective optimization that explicitly trades off latency, cost, and quality. Second, privacy-preserving learning pipelines-combining federated learning with careful distillation-can make on-device models more competent without centralizing user data. Third, edge functionality can evolve from caching and retrieval toward lightweight planning and tool mediation, provided that data minimization and governance remain strict. Finally, standardized evaluation suites for mobile LLM systems should be developed to benchmark end-to-end behavior under realistic network and device heterogeneity. Advancing these areas will help move large-small model collaboration from an engineering workaround to a systematic paradigm for scalable, privacy-respecting mobile intelligence.

References

1. S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
2. P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, and S. Zhao, "Advances and open problems in federated learning," *Foundations and trends® in machine learning*, vol. 14, no. 1-2, pp. 1-210, 2021.
3. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459-9474, 2020.
4. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637-646, 2016. doi: 10.1109/jiot.2016.2579198
5. G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.