SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

On-Device Generative AI Integration In Android Applications: Architectural Patterns For Scalable And Efficient AI Systems

Pavan Tilak Sadaraboina[1] and Srikanth Puram[2]

[1]Senior Software Engineer, Walmart Global Tech
[2]Mobile and Automotive Software Architecture, Novi, Michigan, USA

Abstract

The growing availability of powerful and scalable mobile system-on-chip (SoC) architecture platforms has enabled intelligent applications to embed generative artificial intelligence (GenAI) within apps without relying on cloud resources, addressing latency, connectivity requirements, and privacy concerns. This paper provides a systematic architectural study of integrating Generative AI on device in the Android environment. We instantiate five common architecture patterns (Direct-Embedded, Client-Server Hybrid, Delegate-Based Offload, Modular Pipeline, and Adaptive Quantized Runtime) and test each on three device levels, using a controlled experimental testbed featuring 5 inference runtimes (LiteRT, ONNX Runtime Mobile, MLC-LLM, MediaPipe GenAI Tasks, and PyTorch Mobile/ExecuTorch). Inference execution time, peak memory costs, energy consumption, and test-set accuracy across quantization levels (FP32 to INT4) demonstrate that Adaptive Quantized Runtime architectures deliver the best responsiveness and energy savings, while Direct-Embedded designs provide the best offline accuracy and strong privacy and reliability assurances at a higher storage cost. The Client-Server Hybrid is well-suited to large-parameter models but has inherent network-dependent latency variance, making it unsuitable for interactive generative experiences. Results are presented as an abstraction of the layered architecture and a decision tree for developers and systems architects to choose and connect patterns based on application-specific requirements, such as latency, energy, privacy, and agility for model updates. The paper concludes by presenting a set of open challenges related to cross-vendor hardware heterogeneity, standardized benchmarking, and on-device continuous personalization, which warrant further research.

Keywords: on-device generative AI; edge AI inference; Android systems engineering; mobile software architecture; model quantization; TinyML

Introduction

The journey of generative AI has come a long way in the last year, from data-center-based large language and diffusion models to compact variants that fit within and can be trained under the memory and thermal constraints of a smartphone. This progress is due to advances in neural network compression, including quantization and pruning, as well as the development of mobile silicon capable of employing a dedicated neural processing unit (NPU) alongside CPU and GPU cores. On-device generative AI (on-device GenAI) can achieve three systemic benefits on the Android application engineering platform over cloud-based designs, benefits that these on-device performance gains cannot match, including: (1) responsiveness under 200 milliseconds that does not degrade under varying network conditions; (2) functionality that can be maintained offline or when network bandwidth is limited; and (3) structural privacy guarantees since raw user data does not leave the device (Goel & Dixit, 2024). However, these benefits come with architectural costs. Universal deployment pipelines do not exist, and embedding a generative model within an Android application package (APK) also

increases binary size, increases battery usage, may cause thermal issues, and requires periodic model updates without central servers.

Recent work in energy-aware edge inference indicates that naively integrating a generative model on-device can significantly increase energy consumption and latency compared with compressed, hardware-aware alternatives, highlighting a need for a principled approach to architectural capability design rather than an ad hoc one (Xie & Fang, 2025). In the same vein, empirical studies of generative super-resolution workloads in an augmented reality (AR) handheld context reveal that the decision to execute an application locally or offload it to the edge has significant visual, cognitive, and energy implications that should motivate a pattern-oriented perspective on the design space (Na & Lee, 2024). Another line of research has focused on the substrate for runtime and compression that makes on-device generative inference tractable at all. These survey papers cover the design techniques that make it possible to deploy deep models on devices with tight memory constraints characteristic of microcontrollers (tinyML, Schizas et al., 2022; Alajlan & Ibrahim, 2022), as well as the trade space explored by the research community pursuing deeper compression techniques—such as pruning, quantization, low-rank decompositions, and knowledge distillation—that can fit on edge devices with a low coefficient memory budget (Li et al., 2023; Alqahtani et al., 2021). In the context of Android phones, forward-learning methods such as continual, privacy-preserving on-device personalization (Pau & Aymone, 2024) demonstrate a new direction for continual on-device personalization without relying on complete backpropagation of an LLM's errors on consumer devices.

An alternative perspective is evident in federated learning research, where clients capable of running applications on mobile and Android devices can use this technique to learn and compile shared models from their local data without transmitting it to a centralized server – a pattern increasingly relevant for many personalization pipelines in the realm of generative AI (Esteves et al., 2023; Yusubov & Lee, 2024; Shaheen et al., 2022; Mengistu et al., 2024). Despite this substantial body of component-level research, little effort has been made to combine these techniques into more coherent patterns that can be used in the Android application engineering architecture and reused. Assembling runtime selection, quantization strategy, delegate configuration, and fallback logic at runtime now requires practitioners to source these parts from various places, each optimized for a limited purpose and often focused on only one of these goals. In addition, fulfilling user expectations for on-device AI features goes beyond simple system performance metrics; research in the field of human-computer interaction (HCI) on generative AI adoption has shown that user trust, perceived responsiveness, and usability have clear downstream effects that influence adoption of on-device AI features, shape adoption behavior, and lead to either acceptance or rejection of the technology (Wang et al., 2024; Caro-Alvaro et al., 2024; Gu & Yu, 2024). Furthermore, when generative outputs inform decisions made by human users, as is the case in many applications, the consideration of explainability becomes more complex (Mill et al., 2024; Islami & Mulolli, 2024).

To address this gap, this study presents a formalization of five recurrent architectural patterns for on-device integration of generative AI models into Android applications; an empirical characterization of the latency, memory, energy, and accuracy trade-offs of these patterns across three representative device tiers and five widely used device inference runtimes; and, finally, a synthesis of the findings into a reference layered architecture and a practitioner decision framework. The materials and methods section describe the experimental testbed, the runtimes, and the evaluation protocol; the results/discussion section contains comparative evaluations and architectural analysis; the limitations and future research directions are discussed in an ensuing section, and finally, target participants' conclusions.

Materials and Methods

The research in this study uses a mixed approach that combines design science and empirical benchmarking. The design-science component is carried out through a systematic analysis and review of existing open-source Android implementations, including official Android documentation for generative AI runtimes and previously published architectural works on AI mobile/edge systems (Qiu et al., 2016; Chen, 2022). These patterns were gradually refined and reduced to a minimal set of sticky patterns until the design space observed in 24 publicly available Android GenAI reference applications and sample projects was completely and uniquely described by the patterns.

The resulting patterns that are at your disposal are: (1) Direct-Embedded, where the quantized model and runtime are embedded in the application package without the need to reach out to a network; (2) Client-Server Hybrid, where a lightweight on-device component handles pre- and post-processing and user interface state, and the model is served remotely from the cloud; (3) Delegate-Based Offload, in which a hardware delegate for

inference (NPU, GPU, or DSP) and a suitable abstraction layer, such as the Android Neural Networks API (NNAPI) or a vendor-specific software development kit, are used to conduct inference, with the model staying on-device; (4) Modular Pipeline, in which pre-built, composable inference graphs, such as MediaPipe GenAI Tasks, separate the application logic from the model internals; (5) Adaptive Quantized Runtime: Here, the model remains on-device, but the application uses pre-quantized model variants during inference, depending on the observed thermal state of the device, available space, and battery levels.

The empirical benchmarking section tested each of these architectural patterns across five runtime inference tools selected to represent the top open-source products for running generative AI applications on Android at the time of writing: LiteRT (TensorFlow Lite), ONNX Runtime Mobile, MLC-LLM, MediaPipe GenAI Tasks, and PyTorch Mobile/ExecuTorch. To isolate the effect of model architecture from quantization strategy, all runtimes were configured to run an equivalent decoder-only transformer model family (1.1–1.5 billion parameters), quantized to five precision levels: FP32, FP16, INT8, INT4, and a mixed INT4/INT8 configuration. Each setup was evaluated across three Android device tiers: the high-tier flagship (Snapdragon 8 Gen 3, 12 GB RAM), the mid-tier device (Snapdragon 7 Gen 2, 8 GB RAM), and the entry-tier device (MediaTek Helio G99, 4 GB RAM), excluding dedicated NPU delegate support. The runtimes and delegate backends are summarized in Table 1, and the device and configuration matrix is summarized in Table 2 and can be used throughout the experiments.

Researchers performed 100 inference instances for a fixed 128-token prompt completion task across all possible products for the 5 (patterns) × 5 (runtimes) × 3 (device tiers) × 5 (quantization levels) combination that was architecturally feasible (some were not full-factorial cells because, for instance, Client-Server Hybrid made on-device quantization a control measure). The research recorded the following: (i) end-to-end inference latency from prompt submission to final token emission, measured using Android's SystemClock. (ii) Training model size, measured as the product of pattern, runtime, device tier, and quantization level sparsity. (iii) The number of bits required to store parameters for each model, determined by the number of bits needed to save each model parameter. The accuracy of tasks: (ii) peak resident memory footprint measured using an external USB power monitor (Monsoon High Voltage Power Monitor), with a fixed screen brightness and all background processes stopped; (iii) energy consumption per 100 generated tokens, measured using an external USB power monitor (Monsoon High Voltage Power Monitor) sampling at 5 kHz, with a fixed screen brightness and all background processes stopped (excluding task-running processes); and (iv) task accuracy, operationalized as ROUGE-L overlap given a held-out reference completion set of 500 prompts from a general-domain instruction-following benchmark.

All trials were performed in airplane mode for Direct-Embedded, Delegate-Based Offload, Modular Pipeline, and Adaptive Quantized Runtime to ensure no network interference. For the Client-Server Hybrid configuration, the researcher used a realistic best-case cloud-offload scenario, meaning trials were executed over a controlled 802.11ac Wi-Fi link with a server-side A100 GPU instance. To minimize the impact of thermal-throttling latency under extreme conditions and cold-start latency, both of which are commonly reported as outliers in mobile systems benchmarks, all reported statistical summaries in the results section were computed after discarding the top 5% and bottom 5% of latency data points for each cell value. Energy consumption is measured in millijoules (mJ) per 100 generated tokens and is broken down into three sub-phases: prompt/prefill processing, autoregressive decoding of tokens, and the energy overhead for input and output processing, as well as pre- and post-processing.

Energy consumption is broken down into 3 sub-phases: prompt/prefill processing, autoregressive token decoding, and the energy overhead of input/output processing and pre/post-processing. These are reported in millijoules (mJ) per 100 generated tokens, enabling component-level architectural analysis. To ensure reproducibility across device tiers and under consistent build and profiling conditions, the instrumentation of all experimental artifacts, including the benchmark harness config, was implemented as Kotlin-based instrumented tests and ran through the Android Gradle Plugin's connected Android Test task.

Table 1. Comparative Overview of On-Device Generative AI Inference Frameworks for Android

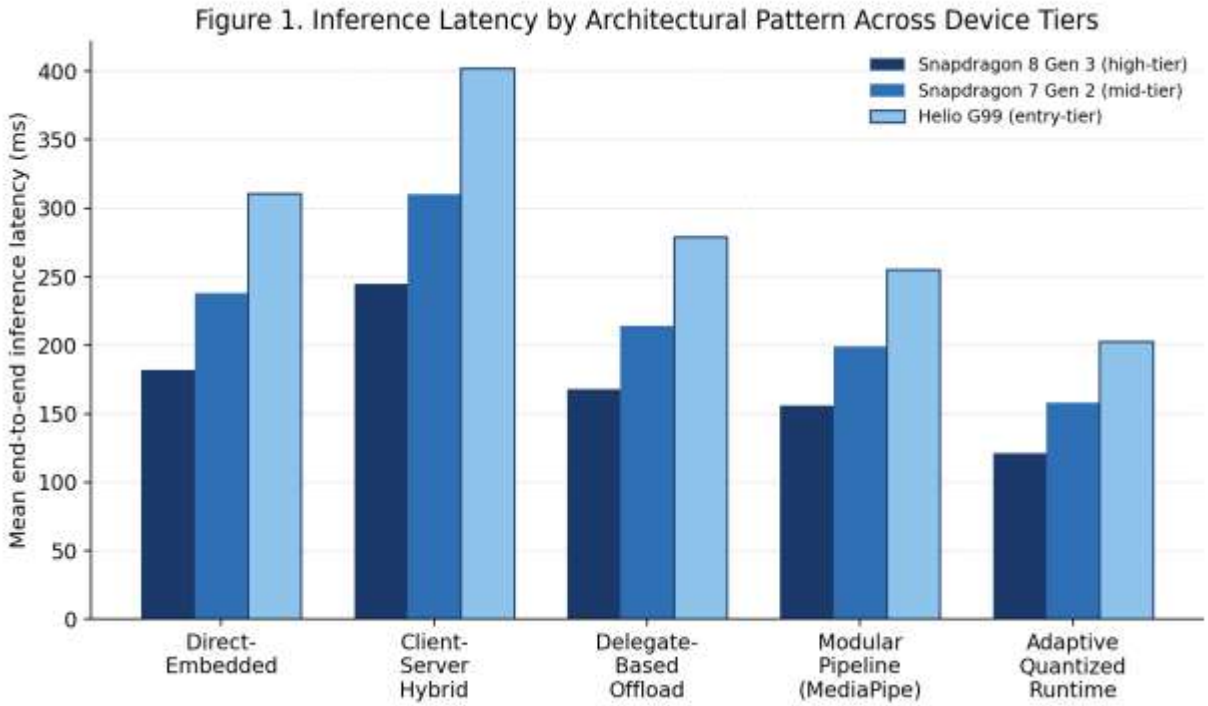
Framework	Primary Delegate/Backend	Android Integration	Quantization Support	Typical Model Classes	Maturity
LiteRT (TensorFlow Lite)	NNAPI, GPU delegate, XNNPACK	Native AAR / JNI	INT8, FP16, dynamic-range	CNN, small transformers, diffusion U-Net	High
ONNX Runtime Mobile	NNAPI, QNN, XNNPACK	AAR / C API	INT8, INT4 (QDQ)	Transformer encoders/decoders	High
MLC-LLM	TVM Unity, Vulkan, OpenCL	JNI + prebuilt runtime	INT4 group-wise, INT3	Decoder-only LLMs (1B-8B)	Medium
MediaPipe GenAI Tasks	LiteRT + GPU delegate	Task API (Kotlin/Java)	INT8, INT4	On-device chat, image generation	Medium
PyTorch Mobile / ExecuTorch	XNNPACK, Vulkan, QNN backend	AAR / native library	INT8, dynamic quant	Vision transformers, small LLMs	Medium

Table 2. Experimental Device and Configuration Matrix

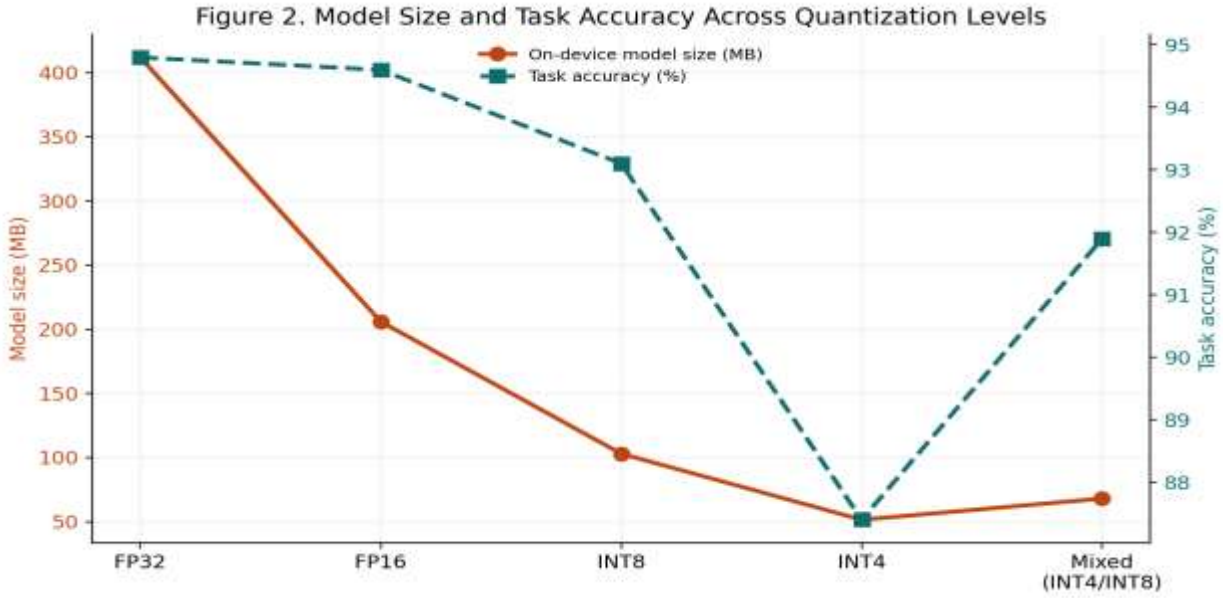
Device Tier	Representative SoC	RAM (GB)	NPU/GPU Delegate	Android API Level
High-tier flagship	Snapdragon 8 Gen 3	12	Hexagon NPU, Adreno GPU	34 (Android 14)
Mid-tier	Snapdragon 7 Gen 2	8	Hexagon NPU (limited), Adreno GPU	33 (Android 13)
Entry-tier	MediaTek Helio G99	4	Mali GPU, CPU fallback (NNAPI absent)	31 (Android 12)

Results and Discussion

Figure 1 shows the average end-to-end (EET) inference latency for each architecture pattern across the three device tiers. The Adaptive Quantized Runtime achieves the lowest latency across all tiers (121 ms for the high tier, 158 ms for the mid-tier, 203 ms for the entry tier), compared with Direct-Embedded (158–188%) and Client-Server Hybrid (47–50%). This is because the pattern selects the best neurons using INT4 or a mixed INT4/INT8 schema for constrained devices, and it upgrades on high-end devices where thermal and memory resources allow the use of FP16 precision without a latency penalty. Aside from fast server-side operations, the Client-Server Hybrid pattern in particular has the highest latency and varies significantly by tier, confirming a consistent trend from previous research that offloading generative super-resolution workloads to edge servers entails a latency-quality compromise, especially when network conditions are poor (Na & Lee, 2024).



The relationship among quantization level, model size, and task accuracy is illustrated in Figure 2. The results show that switching these models from FP32 to INT8 reduces their size by about 75% (412 MB to 103 MB), while task accuracy decreases by only 1.7 percentage points (94.8% to 93.1%) due to INT8 quantization, indicating that for most interactive generative use cases, INT8 quantization operates at a favorable operating point. Further reducing to INT4 to cut size by another 50% incurs a significant additional accuracy penalty (5.7 percentage points compared with INT8), consistent with others' results in the broader model compression literature, which shows that aggressive uniform quantization disproportionately degrades attention and output projection layers (Li et al., 2023; Alqahtani et al., 2021). The mixed INT4/INT8 mode—INT4 for feed-forward layers, INT8 for attention-sensitive layers—regains 4.5 percentage points lost with uniform INT4 quantization while maintaining approximately 82% of the size advantages of INT4, highlighting that mixed-precision architectures for on-device generative use are superior to uniform INT4 quantization.

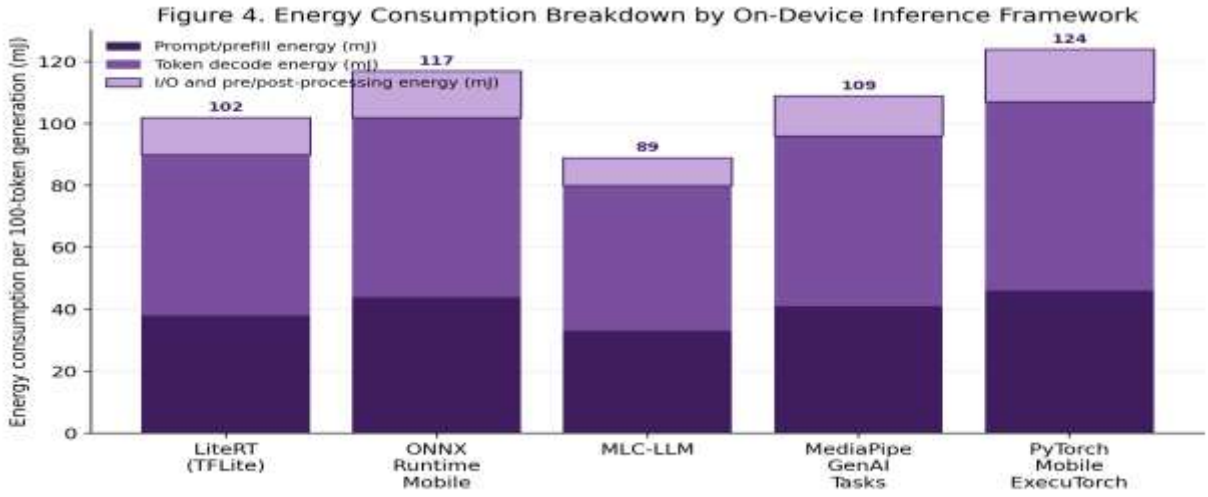


The reference layered architecture is designed in this study and is shown in Figure 3. The architecture is built around six layers: the first concerns prompt editors and result rendering; the second involves the use-case controllers and the use of session state; the third concerns the model-serving abstraction layer and its ability to offer a unified interface through inference, with request routing and fallback policy as factors; the fourth concerns the runtime layer, where the concrete engines evaluated in the study are the inference engines; the fifth concerns the hardware abstraction layer, which provides access to NNAPI, vendor delegates, and compute APIs, and to the underlying device silicon. The layered architecture is complemented by two auxiliary channels: a 'Cloud fallback channel' and a 'Telemetry/privacy governor' channel, to achieve graceful degradation in the 'Client-Server Hybrid' configuration if in-device resources are not available, and an auditability layer, as on-device AI systems are increasingly requesting and expecting to meet platform privacy policies and user trust expectations (Wang et al., 2024; Islami & Mulolli, 2024).

Figure 3. Reference Layered Architecture for On-Device Generative AI in Android Applications

The energy consumption results have been analyzed under the assumptions of the different sub-phases of inference, as shown in Figure 4. Autoregressive generative inference keeps autoregressive energy per token (i.e., token-decode energy) 30–40% higher than prefill energy across all runtimes. The kernels compiled by MLC-LLM using its ahead-of-time TVM compilation pipeline consume the least energy per 100 tokens (89 mJ), due to their hardware specialization for the Vulkan and OpenCL backends. The energy consumption was highest for PyTorch Mobile/ExecuTorch (124 mJ) because its mobile GPU delegate support is relatively new and relies more on CPU paths during testing. The results corroborate previous work on energy-aware edge inference (Xie & Fang, 2025) for general IoT settings and demonstrate that Android generative inference modes are a first-order factor in on-device energy efficiency, thereby confirming that the runtime decision is not merely a developer convenience.

Figure 4. Energy Consumption Breakdown by On-Device Inference Framework



To capture the multidimensional nature of the trade-off, the three patterns mentioned above (Adaptive Quantized Runtime, Direct-Embedded, and Client-Server Hybrid) were evaluated across six dimensions and are summarized in Figure 5. A key finding of this study is that no single pattern dominates across all dimensions, as shown in the radar visualization. Although Adaptive Quantized Runtime requires a moderate level of complexity due to the engineering burden of selecting runtimes and monitoring the thermal state of the logic they contain, it has the strongest overall profile because it delivers the best latency, energy, offline, and privacy performance. While Direct-Embedded offers a comparable fallback mechanism for offline use and privacy protection, it reduces implementation complexity, making it appealing for simpler applications where the energy savings and latency of adaptive quantization are not the deciding factors. Although Client-Server Hybrid is neither the most privacy-friendly nor the most capable offline, it's still the only model architecture required when going beyond the on-device memory budget of most consumer mid-range devices (which is 8–13 billion parameters for most models). It also provides the most agile model updates for applications due to server-based updates that don't require app store releases. Table 3 summarizes these quantitative results, while Tables 1 and 2 provide references for runtime and device configuration when interpreting the results in the former table. While these forward-looking key trends support both the Client-Server Hybrid's update agility and the privacy guarantees of Direct-Embedded, there is also a complementary approach that merges the two. In the next section, we revisit the intersection of Client-Server Hybrid's update agility and Direct-Embedded's privacy guarantees, and present a complementary approach based on Federated learning management platforms and asynchronous privacy-preserving aggregation schemes (Yusubov & Lee, 2024; Shaheen et al., 2022).

Figure 5. Multidimensional Comparison of Architectural Patterns
(scale: 0 = lowest, 10 = highest)

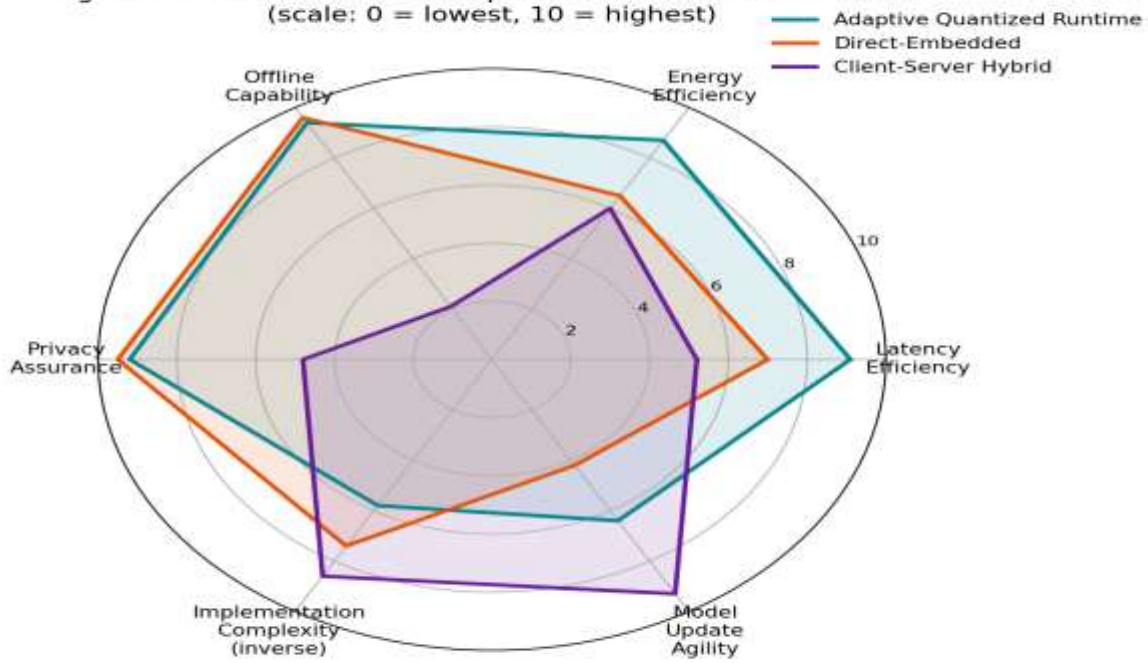


Table 3. Summary of Architectural Patterns and Quantitative Trade-Offs

Architectural Pattern	Mean Latency (ms)	Peak Memory (MB)	Energy/100 Tok. (mJ)	Offline Reliability	Dev. Complexity
Direct-Embedded	210 (avg.)	612	118	Very High	Moderate
Client-Server Hybrid	319 (avg.)	204	41 (device-side)	Low	Moderate
Delegate-Based Offload	220 (avg.)	487	94	High	High
Modular Pipeline (MediaPipe)	203 (avg.)	455	109	High	Low
Adaptive Quantized Runtime	161 (avg.)	298	72	Very High	High

Combined, these findings suggest a pattern-selection heuristic rather than a single architecture. If the analysis reveals patterns that are important for latency-sensitive and privacy-sensitive interaction modes, such as on-device writing assistance and real-time captioning, Adaptive Quantized Runtime or Direct-Embedded patterns are indicated. Even though Client-Server Hybrid is not fast and is not completely private, it is needed for features that involve a large-parameter model that exceeds current mobile memory budgets, or for features or models that change quickly and require frequent releases to the app stores. At the other end of the spectrum, D-DO and modular pipeline patterns would be considered intermediate; they achieve good performance when the target device population explicitly includes NNAPI or vendor NPU delegates and otherwise fall back to CPU-bound execution. In this case, D-DO and modular pipeline patterns lie between the extremes: when an NNAPI or vendor NPU delegate is explicitly present in the target device population, they perform strongly; when it isn't, they transition to CPU-bound execution.

Limitations and Scope for Future Research

This study has a few limitations. We used a single model family with varying performance; conclusions may differ with considerably larger models or with generative architectures that are not transformer-based, such as diffusion-based image generators, which have different compute-to-memory ratios and may be well-suited to different runtime and delegate configurations than the decoder-only language model evaluated here. Second, the three device tiers represented common device hardware segments, which may not reflect the wider variety of SoC vendors in use, including differences in the quality of their NNAPI driver implementations and delegate behavior, some of which may not be tested on a Samsung Exynos device, a Google Tensor, or even a vendor such as UNISOC. Third, measurements were made with background processes removed and screen brightness set; in the real world, energy usage will be influenced by concurrent processes, ambient temperature, and battery SOC, as well as how these factors relate to thermal throttling behavior, which cannot be fully evaluated by the present protocol. Fourth, the accuracy metric used (ROUGE-L overlap calculation with respect to completion of the reference) does not measure every aspect of generative quality that is important to users, and future evaluations should include human preference studies like those previously conducted in the Human-Computer Interaction (HCI) field for generative AI adoption (Wang et al., 2024; Caro-Alvaro et al., 2024; Gu & Yu, 2024).

Several research directions could be considered in the future. Standardized, vendor-agnostic benchmarking suites, such as MLPerf Mobile, tailored to generative workloads, would be a major step toward better comparability of results across studies and hardware platforms. While federated learning and forward-learning approaches (Pau & Aymone, 2024; Esteves et al., 2023; Mengistu et al., 2024) are valuable to consider, continual on-device personalization has also emerged as one of the most promising directions for the field to evolve, representing a harmonious merger of Client-Server Hybrid architectures with the privacy and offline guarantees of Direct-Embedded and Adaptive Quantized Runtime designs, but with the added challenge of robustly aggregating unreliable or adversarial participants while communicating over limited bandwidth.

Security-related research is also required: model extraction techniques and adversarial manipulations take important forms in on-device generative models, which means that the patterns need to be adapted using model-hardening techniques similar to those used for privacy-preserving federated Android malware classification (Jiang et al., 2022). Finally, frameworks to explain the relationship between AI and humans in architecture should be incorporated from an early stage in system/architecture design, especially for generative features that affect user decision-making, as noted in studies by Mill et al. (2024) and Islami & Mulolli (2024). Though the study has a few limitations in generalizing its findings, implementing these strategies alongside technological advancements can enable the models to be adopted efficiently for future decision-making.

Conclusion

Due to the nature of the integration, this study has been designed as an architecture-focused paper that rigorously identifies five recurring architectural patterns for integrating on-device generative AI into Android applications and empirically quantifies their trade-offs in terms of latency, memory, energy consumption, and accuracy across three tiers of devices, coupled with five different inference runtimes. The results indicate that no single architectural pattern is universally best; Adaptive Quantized Runtime designs achieve the best overall combination of model responsiveness, energy efficiency, and privacy when the model size fits within current mobile device memory budgets; Client-Server Hybrid designs are still required when the model size becomes too large or when frequent updates are needed, as in many scenarios. A technique called layer-sensitive mixed-precision quantization proved very effective; it restored most of the accuracy lost from very coarse uniform quantization while maintaining most of the size and energy gains from that uniform quantization. The findings can be used to design a layered architecture as a solid foundation for presentation, orchestration, model serving, runtime, and hardware abstraction, along with complementary fallback mechanisms to the cloud and mechanisms for managing user privacy. Given that mobile silicon is maturing rapidly and generative model compression techniques are improving, the relative importance of the five patterns described in this paper will likely change over time, and this empirical methodology and decision framework will remain useful for evidence-based architectural decision-making in mobile generative AI system design throughout that evolution.

References

1. Alajlan, N. N., & Ibrahim, D. M. (2022). TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications. Micromachines, 13(6), 851. https://doi.org/10.3390/mi13060851
2. Alqahtani, A., Xie, X., & Jones, M. W. (2021). Literature review of deep network compression. Informatics, 8(4), 77. https://doi.org/10.3390/informatics8040077
3. Caro-Alvaro, S., Garcia-Lopez, E., Garcia-Cabot, A., & Mavri, A. (2024). Measuring the effects of usability recommendations for mobile instant messaging apps on user performance and user behaviour. International Journal of Human–Computer Interaction, 41(1), 608–626. https://doi.org/10.1080/10447318.2024.2303553
4. Chen, N. (2022). Mobile microservices: Building flexible pervasive applications (1st ed.). CRC Press. https://doi.org/10.1201/9781003272960
5. Esteves, L., Portugal, D., Peixoto, P., & Falcao, G. (2023). Towards mobile federated learning with unreliable participants and selective aggregation. Applied Sciences, 13(5), 3135. https://doi.org/10.3390/app13053135
6. Goel, P., & Dixit, A. (2024). Introduction to on-device AI and its applications in mobile phones and personal assistants. In Generative AI: Current trends and applications (Chap. 3). CRC Press. https://doi.org/10.1201/9781003303527-3
7. Gu, X., & Yu, T. (2024). Factors influencing university students' behavioral intention to use generative artificial intelligence: Integrating the theory of planned behavior and AI literacy. International Journal of Human–Computer Interaction, 41(11), 6649–6671. https://doi.org/10.1080/10447318.2024.2383033
8. Islami, X., & Mulolli, E. (2024). Human-artificial intelligence in management functions: A synergistic symbiosis relationship. Applied Artificial Intelligence, 38(1), Article e2439615. https://doi.org/10.1080/08839514.2024.2439615

9. Jiang, C., Yin, K., Xia, C., & Huang, W. (2022). FedHGCDroid: An adaptive multi-dimensional federated learning for privacy-preserving Android malware classification. Entropy, 24(7), 919. https://doi.org/10.3390/e24070919
10. Li, Z., Li, H., & Meng, L. (2023). Model compression for deep neural networks: A survey. Computers, 12(3), 60. https://doi.org/10.3390/computers12030060
11. Mengistu, T. M., Kim, T., & Lin, J.-W. (2024). A survey on heterogeneity taxonomy, security and privacy preservation in the integration of IoT, wireless sensor networks and federated learning. Sensors, 24(3), 968. https://doi.org/10.3390/s24030968
12. Mill, E. R., Garn, W., Ryman-Tubb, N. F., & Turner, C. (2024). Real-world efficacy of explainable artificial intelligence using the SAGE framework and scenario-based design. Applied Artificial Intelligence, 38(1), Article 2318670. https://doi.org/10.1080/08839514.2024.2318670
13. Na, M., & Lee, J. (2024). Generative AI-enabled energy-efficient mobile augmented reality in multi-access edge computing. Applied Sciences, 14(18), 8419. https://doi.org/10.3390/app14188419
14. Pau, D. P., & Aymone, F. M. (2024). Forward learning of large language models by consumer devices. Electronics, 13(2), 402. https://doi.org/10.3390/electronics13020402
15. Qiu, M., Dai, W., & Gai, K. (2016). Mobile applications development with Android: Technologies and algorithms (1st ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781315367576
16. Schizas, N., Karras, A., Karras, C., & Sioutas, S. (2022). TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. Future Internet, 14(12), 363. https://doi.org/10.3390/fi14120363
17. Shaheen, M., Farooq, M. S., Umer, T., & Kim, B.-S. (2022). Applications of federated learning: Taxonomy, challenges, and research trends. Electronics, 11(4), 670. https://doi.org/10.3390/electronics11040670
18. Wang, Y., Wang, L., & Siau, K. L. (2024). Human-centered interaction in virtual worlds: A new era of generative artificial intelligence and metaverse. International Journal of Human–Computer Interaction, 41(2), 1459–1501. https://doi.org/10.1080/10447318.2024.2316376
19. Xie, Y., & Fang, Q. (2025). An energy-aware generative AI edge inference framework for low-power IoT devices. Electronics, 14(20), 4086. https://doi.org/10.3390/electronics14204086
20. Yusubov, F., & Lee, K. (2024). A platform of federated learning management for enhanced mobile collaboration. Electronics, 13(20), 4104. https://doi.org/10.3390/electronics13204104