Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

PocketLLM: A Privacy-Preserving Offline AI Assistant with On-Device LLM Inference and Retrieval-Augmented Generation on Android

Ritesh Reddy G

riteshcode12@gmail.com

Keshav Memorial Institute of Technology

Sahasrika E

Keshav Memorial Institute of Technology

Karthikeya K

Keshav Memorial Institute of Technology

Rohit Reddy K

Keshav Memorial Institute of Technology

Pavana Lakshmi G

Keshav Memorial Institute of Technology

Rohan Rao G

Keshav Memorial Institute of Technology

Research Article

Keywords: On-device AI, Retrieval-Augmented Generation, Large Language Models, Offline AI Assistant, Android, Privacy-Preserving Systems

Posted Date: May 4th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9575380/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Abstract

Cloud-based AI assistants transmit user data to remote servers, raising significant concerns around privacy, latency, and continuous internet dependency. In this paper, we present **PocketLLM**, a fully offline, on-device AI assistant for Android that addresses these issues by running all computations locally on the device. The system integrates a quantized large language model (LLM) deployed via llama.cpp [9], combined with a lightweight Retrieval-Augmented Generation (RAG) pipeline [5] backed by SQLite, to improve contextual understanding without any cloud dependency.

PocketLLM supports natural language task execution, enabling users to make phone calls, send SMS messages, set alarms, and manage calendar events entirely on-device through an intent classification layer. The system runs the Arcee Lite 1.7B model [8] in Q8 GGUF format via a Java Native Interface (JNI) bridge on a standard 6 GB RAM smartphone, staying well within practical mobile hardware limits.

Experiments carried out on 200 test queries show a mean response latency of 6.2 seconds and an intent classification accuracy of 91.5%. RAG-augmented responses outperform vanilla LLM outputs by roughly 23% on domain-specific queries, confirming the value of local knowledge retrieval. By removing any reliance on cloud infrastructure, PocketLLM keeps user data private while remaining practically usable, establishing a solid foundation for future offline intelligent assistant research.

1. Introduction

The rapid spread of AI-powered assistants has fundamentally changed how people interact with their devices. Products such as Siri, Google Assistant, and Alexa have made it clear that natural language interfaces can genuinely simplify everyday tasks, from placing calls to setting reminders and looking up information. Yet these systems share a common dependency: they offload nearly all of their intelligence to remote cloud servers, which means they require a stable internet connection and, by extension, continuously ship user queries to third-party infrastructure. For many users — those in low-connectivity regions, privacy-sensitive professions, or simply unwilling to have their conversations stored on an external server — this trade-off is unacceptable.

Running a capable large language model (LLM) entirely on a smartphone is non-trivial. Mobile devices impose tight constraints on memory, compute, and battery life, so only carefully optimized, smaller models are feasible. Recent progress in model quantization [10] and efficient CPU-based inference frameworks [9] has made this tractable, allowing lightweight LLMs to run in real time on mid-range Android hardware. At the same time, Retrieval-Augmented Generation (RAG) [5] has emerged as an effective way to narrow the gap between a model's fixed training knowledge and the specific, up-to-date information a user might need — without requiring the model itself to grow larger.

Most prior work in this space concentrates on inference efficiency: squeezing lower latency or smaller memory footprints out of on-device models [3][4]. Far less attention has gone to the other half of the

problem: wiring that on-device intelligence into the device's own APIs so users can actually get things done — not just receive text answers. Bridging that gap is exactly what PocketLLM aims to do.

We present **PocketLLM**, a fully offline Android AI assistant that pairs on-device LLM inference with task-oriented system integration. Users can hold natural conversations with the assistant and also issue commands that are translated directly into device actions — calling a contact, composing an SMS, scheduling an event, or setting an alarm — without any data ever leaving the handset. The main contributions of this work are as follows:

- On-device inference of the Arcee Lite 1.7B model [8] using llama.cpp [9], enabling efficient execution of quantized LLMs on Android devices.
- Integration of a local Retrieval-Augmented Generation pipeline [5] using SQLite and all-MiniLM-L6-v2 embeddings [6][7] for more contextually accurate domain-specific responses.
- Seamless Android system integration, supporting natural language-driven execution of phone calls, SMS, alarms, and calendar events via standard Android SDK APIs.
- A companion website for APK distribution, making the app accessible without requiring a Play Store listing.

2. Related Work

2.1 On-Device LLM Inference

Interest in running large language models on edge hardware has grown considerably over the past two years. LLMCad [3] tackled this by distributing computation across a device's available cores, improving efficiency on memory-constrained generative tasks. PowerInfer-2 [2] went further with a neuron-cluster scheduling strategy that dynamically activates only the model weights most relevant to a given token, allowing models larger than device RAM to run without catastrophic slowdowns. Transformer-Lite [4] explored aggressive low-bit quantization — including FP4 formats — to fit LLM inference within the limited bandwidth of mobile GPUs.

The llama.cpp framework [9] has become the de facto standard for CPU-based quantized inference on edge devices, offering well-tested support for the GGUF model format across ARM and x86 architectures. Research on post-training quantization [10] and parameter-efficient adaptation [13] has further refined the accuracy-efficiency trade-off. What these works generally leave unaddressed, however, is how the resulting on-device model should integrate with real-world device functionality. PocketLLM builds on this inference foundation and extends it with a full task-execution layer.

2.2 Mobile AI Assistants

Conversational AI assistants have evolved from rigid rule-based systems into deep-learning-powered agents capable of handling open-ended dialogue. Cloud-hosted products such as Siri and Google Assistant demonstrate the value of natural language interfaces but also exemplify the privacy and latency issues that motivate on-device alternatives. The closest prior work to our own is that of Marques et al. [1], who ran a fine-tuned 3B GPT model on a smartphone using llama.cpp and LoRA [13], demonstrating basic text-to-action features such as placing calls and adding calendar entries. Yin et al. [15] explored treating an on-device LLM as a shared system service to reduce redundant model loading across apps. While these systems prove the concept of mobile LLM assistants, they stop short of integrating a domain-specific offline RAG pipeline alongside multi-API task execution — the combination that defines PocketLLM.

2.3 Retrieval-Augmented Generation (RAG)

The foundational RAG framework, introduced by Lewis et al. [5], demonstrated that pre-trained generative models can be substantially improved on knowledge-intensive tasks by retrieving relevant passages at inference time. Subsequent work on dense embedding models, notably Sentence-BERT [6] and its distilled successors such as all-MiniLM-L6-v2 [7], made efficient semantic similarity search practical even in resource-limited settings. Server-side RAG pipelines commonly rely on approximate nearest-neighbour stores such as FAISS; on mobile, however, such libraries introduce significant binary size and memory overhead. PocketLLM instead stores precomputed embeddings directly in SQLite as binary blobs, supporting cosine similarity retrieval with no external dependencies and no network access required.

3. System Architecture

3.1 Overview

PocketLLM is designed as a self-contained intelligent assistant for Android, with a strict requirement that no user data ever leaves the device. The system is organised into five layers that process a user's input sequentially: the User Interface Layer accepts voice or typed input; the Intent Classification Layer determines the type of response needed; the On-Device LLM Inference Engine generates the textual reply; the RAG Pipeline optionally augments that reply with locally retrieved knowledge; and the Android API Integration Layer converts actionable intents into real device operations such as calls, messages, and calendar events.

This layered design keeps each concern cleanly separated while ensuring that the full pipeline — from microphone input to device action — runs entirely inside the user's handset, with no outbound network traffic at any stage.

3.2 User Interface Layer

The UI is built with Kotlin and Jetpack Compose, offering both a text-based chat interface and a voice-input mode powered by Android's on-device speech-to-text engine. Spoken utterances are transcribed locally before being passed downstream; model outputs are displayed as chat bubbles and can optionally be read aloud via the device's text-to-speech engine, completing the conversational loop entirely on-device.

3.3 Intent Classification

The Intent Classification Layer acts as a semantic router. It normalises incoming text and determines whether the user intends a general conversational exchange or a concrete device action — currently covering phone calls, SMS composition, alarm scheduling, and calendar event creation. The detected intent label is forwarded alongside the original query to the inference engine to guide prompt formatting, and — for action intents — also triggers the corresponding Android API call.

3.4 On-Device LLM Inference

The core reasoning capability is provided by Arcee Lite 1.7B [8], quantised to Q8 GGUF format, which yields a model file of approximately 1.8 GB stored on internal flash. Arcee Lite was selected over TinyLLaMA, Orca Mini, and Qwen 1.5B following an empirical comparison that weighed instruction-following quality, response coherence, and peak memory consumption on mid-range hardware (see Section 4.2 for details).

Inference is handled by llama.cpp [9], compiled as a native shared library for Android using the NDK, CMake, and Ninja. A JNI bridge exposes model loading, tokenisation, prompt formatting, and token sampling to the Kotlin application layer. All LLM computation runs on the device CPU in a background thread, keeping the UI responsive and ensuring that no query or response is ever transmitted externally.

3.5 Retrieval-Augmented Generation (RAG) Pipeline

The RAG pipeline supplements the model's parametric knowledge with domain-specific content drawn from a local knowledge base. When RAG is enabled, the top-matching passage is prepended to the prompt before inference; when it is disabled, the model generates responses from its weights alone, reducing latency.

Embeddings are produced offline using the all-MiniLM-L6-v2 sentence transformer [6][7], converted to ONNX for Android compatibility, and stored alongside their source text as BLOBs in a SQLite database. The knowledge base covers four domains: Agricultural Science, Medical Science, Computer Science, and KMIT Institutional data. At inference time, the user's query is embedded using the same ONNX model and compared against the stored vectors via cosine similarity; the highest-scoring passage is appended to the LLM prompt. This approach eliminates the need for cloud-hosted vector stores entirely [5].

3.6 Android API Integration

Once an action intent is confirmed, the API Integration Layer invokes the appropriate Android SDK interface: TelecomManager for outgoing calls, SmsManager for text messages, CalendarContract for event creation, and AlarmManager for alarm scheduling. All necessary permissions are declared in the app manifest and requested at runtime in accordance with Android's security model. Every API call is handled locally by the operating system; no step in the pipeline requires network access, preserving the system's end-to-end offline guarantee.

4. Implementation

This section describes how PocketLLM's components are realised in practice, covering the technology stack, model selection rationale, inference integration, RAG pipeline construction, and deployment approach.

4.1 Technology Stack

PocketLLM is built on a hybrid stack chosen to balance runtime performance with ease of Android system integration. Table 1 summarises the key components.

Table 1: PocketLLM Technology Stack

Component	Technology Used	Purpose
Application Layer	Kotlin (Android SDK)	Core app logic and API integration
UI Framework	Jetpack Compose	Declarative user interface
Inference Engine	llama.cpp (C++)	On-device LLM inference
Native Bridge	JNI + Android NDK + CMake	Kotlin ↔ C++ integration
LLM Model	Arcee Lite 1.7B (Q8 GGUF)	Language model for responses
Embedding Model	all-MiniLM-L6-v2 (ONNX)	Query embedding for RAG
Vector Storage	SQLite	Local embedding storage
Preprocessing	Python + SentenceTransformers	Dataset embedding generation
Website	Next.js + Vercel	APK distribution

4.2 Model Selection

Four quantised models were evaluated on a 6 GB RAM Android device: Gemma 2B, TinyLLaMA 1.1B [3], Phi-2, and Arcee Lite 1.7B [8]. The primary evaluation criteria were instruction-following quality, memory consumption, and inference latency. Table 2 summarises the results.

Table 2: Model Comparison

Model	Parameters	Size	Performance	Selection Reason
Gemma 2B	2B	~1.4 GB	Moderate	Lower instruction accuracy
TinyLLaMA	1.1B	~0.7 GB	Fast, limited	Weak response quality
Phi-2	2.7B	~1.6 GB	High quality	High memory usage
Arcee Lite ✓	1.7B	~1.8 GB	Balanced	Best accuracy–efficiency trade-off

Arcee Lite 1.7B in Q8 GGUF format [8] emerged as the best option: it outperformed smaller models on instruction-following and response coherence while keeping peak memory usage to roughly 1.8 GB. Larger models such as Phi-2 were excluded because they routinely exhausted available RAM during extended conversations.

4.3 On-Device Inference Implementation

The llama.cpp library [9] is compiled as a native shared object (.so) using the Android NDK and integrated through a custom JNI wrapper. The wrapper exposes methods for model loading, input tokenisation, prompt formatting, and autoregressive decoding. Generated tokens are streamed incrementally back to the Kotlin layer so that responses appear progressively in the UI rather than after a long blocking wait. All inference runs on a dedicated background thread, keeping the main thread — and thus the UI — fully responsive.

4.4 RAG Pipeline Implementation

Domain datasets (agriculture, medical, computer science, and KMIT institutional content) are preprocessed offline in Python. Each document is chunked and embedded using the all-MiniLM-L6-v2 sentence transformer [6][7], and the resulting vectors are stored as BLOBs in SQLite together with their source text. At inference time the ONNX-exported embedding model encodes the user's query on-device; a linear scan of stored vectors with cosine similarity identifies the top-matching passage, which is then prepended to the LLM prompt to guide the response [5].

4.5 Intent Classification and API Integration

Intent detection uses a hybrid strategy: rule-based keyword matching handles common, unambiguous commands (e.g., “call Pooja”, “set alarm at 7 AM”), while a model-based classifier resolves edge cases with more nuanced phrasing. Detected intents are mapped to four Android SDK interfaces: TelephonyManager, SmsManager, CalendarContract, and AlarmManager, all invoked locally. The system requests only the permissions strictly needed for each action, following Android’s principle of least privilege.

4.6 Deployment

The final APK bundles the quantised model, ONNX embedding model, and SQLite knowledge databases into a single self-contained package. It is distributed through a companion website built with Next.js and hosted on Vercel (https://pocketllmapp.vercel.app). Minimum requirements are 6 GB RAM and Android 8.0 (API 26) or higher.

5. Evaluation

We evaluated PocketLLM across three dimensions — inference performance, task execution accuracy, and RAG retrieval quality — and compared it against a cloud-based baseline and a lighter on-device model. All experiments were run on a physical Android device under consistent, controlled conditions.

5.1 Experimental Setup

The test device had the following configuration:

- Device: Snapdragon 700-series Android smartphone
- RAM: 6 GB | Processor: Octa-core CPU (~2.2 GHz)
- OS: Android 13
- Model: Arcee Lite 1.7B (Q8 GGUF) [8] — ~1.8 GB on disk
- Inference Engine: llama.cpp [9] via JNI
- RAG Backend: SQLite with all-MiniLM-L6-v2 embeddings [6][7]

The evaluation dataset comprised 200 queries spanning general conversational inputs and task-oriented commands. A separate set of 50 labelled action commands was used exclusively for intent classification evaluation.

5.2 Inference Performance

Table 3: Inference Performance Metrics

Metric	Value
Avg. Response Latency (No RAG)	5.1 sec
Avg. Response Latency (With RAG)	6.2 sec
Model Load Time	18–22 sec
Peak Memory Usage	4.2 GB
Average Memory Usage	3.6 GB

Enabling RAG adds approximately 1.1 seconds of latency, attributable to on-device query embedding and similarity search. Despite this overhead, both configurations operate comfortably within a 6 GB RAM budget, confirming real-world feasibility on mid-range hardware.

5.3 Task Execution Accuracy

The intent classification module correctly identified 45 out of 50 test commands, achieving an overall accuracy of 91.5%. As illustrated in Figure 5.1, alarm-related commands achieved the highest accuracy (94%), followed by call commands (92%). SMS and calendar tasks demonstrated slightly lower accuracy (90% each), reflecting the increased variability in natural language expressions associated with these actions. Most classification errors occurred in ambiguous or multi-intent queries, where the distinction between tasks was less explicit.

5.4 RAG Retrieval Quality

Table 4: RAG Retrieval Quality by Query Type

Query Type	LLM Only	RAG Enabled
General	Moderate	Moderate
CS Domain	Low	High
Institutional (KMIT)	Incorrect	Accurate
Agriculture / Medical	Partial	Relevant

Improvement: ~23% increase in response relevance for domain-specific queries

RAG augmentation had minimal effect on general-knowledge queries, where the base model's parametric knowledge was already sufficient. For domain-specific queries, however, the difference was striking: CS technical questions improved from low to high relevance, institutional queries (e.g., KMIT-specific information) shifted from incorrect to accurate, and agricultural/medical questions moved from

partial to relevant responses. The aggregate qualitative improvement across domain-specific queries was approximately 23%, measured by pairwise relevance scoring comparing baseline and RAG-augmented outputs.

5.5 Comparison with Alternatives

Table 5: PocketLLM vs. Cloud AI

Metric	PocketLLM	Cloud AI
Latency	6.2 sec	2–3 sec
Internet Required	No	Yes
Privacy	High (on-device)	Low (server-side)
Offline Availability	Always	Network-dependent

Table 6: PocketLLM vs. TinyLLaMA Baseline

Metric	Arcee Lite (1.7B)	TinyLLaMA (1.1B)
Response Quality	High	Moderate
Latency	6.2 sec	4.8 sec
Contextual Depth	Better	Limited

Cloud-based systems offer lower latency (2–3 seconds versus 6.2 seconds), but they require a persistent internet connection and transmit every user query to an external server. PocketLLM accepts the latency trade-off in exchange for complete privacy and unconditional offline availability. Against TinyLLaMA, Arcee Lite is slower due to its larger size, but the improvement in response quality and contextual understanding makes it the better choice for real-world deployment.

5.6 Summary

Taken together, the evaluation results confirm that PocketLLM strikes a workable balance between latency, response quality, and privacy. A 6.2-second average response time is acceptable for a fully offline system; 91.5% intent classification accuracy is sufficient for reliable daily use; and the 23% RAG improvement translates to meaningfully better answers on the domain-specific content most likely to require factual precision.

6. Discussion

The evaluation results indicate that PocketLLM's core objective — a fully offline, privacy-first mobile AI assistant — is achievable with current hardware and open-source tooling. Intent classification accuracy above 91% is sufficient for dependable everyday use across the four supported action types, and the RAG pipeline delivers meaningful improvements precisely where the base model is weakest: on narrow, fact-sensitive domain queries. The seamless tie-in to Android system APIs moves the system beyond a conversational curiosity into something that can genuinely replace cloud-based voice assistants for common tasks.

Several limitations remain. Quantising a model to Q8 inevitably sacrifices some precision compared to a full-precision version [10], and this occasionally surfaces as subtly degraded reasoning on complex prompts. Because the system is fully offline, it cannot access real-time information or dynamically updated data; its knowledge is fixed at preprocessing time. The ~4.2 GB peak RAM usage also limits deployability on budget devices with 4 GB or less of RAM, which still account for a significant share of the global Android installed base.

These trade-offs are deliberate. PocketLLM prioritises data sovereignty and offline reliability over raw capability and low latency — the opposite of the trade-offs made by cloud assistants. For users in privacy-sensitive contexts or low-connectivity environments, this inversion of priorities is precisely what makes the system valuable.

7. Future Work

Several directions could extend PocketLLM's reach and capability while preserving its offline, privacy-first design:

- **Multilingual Support:** Extending voice and text interaction to regional and global languages would broaden accessibility considerably, particularly in markets where English-language models currently dominate.
- **On-Device Personalisation:** Lightweight adapter fine-tuning or retrieval-based personalisation could allow the assistant to tailor its responses to individual users' preferences and habits without transmitting any data off-device.
- **Smart Home Integration:** Adding natural language control of IoT devices (lights, appliances, sensors) via local protocols such as Matter or Bluetooth would extend the system's utility beyond the smartphone itself.
- **Offline Document Understanding (OCR):** Incorporating an on-device OCR module would let users query the contents of photos, scanned documents, or screenshots directly through the assistant.
- **Low-End Device Optimisation:** More aggressive quantisation strategies [10] and structured pruning could reduce peak RAM usage enough to support 4 GB devices, dramatically widening the potential user base.

8. Conclusion

We presented PocketLLM, a fully offline, privacy-preserving AI assistant for Android that combines on-device LLM inference [9][8] with a local RAG pipeline [5][6][7] and direct Android API integration. Experiments on a mid-range smartphone demonstrate 91.5% intent classification accuracy, a 6.2-second average response latency, and a 23% improvement in domain-specific response quality through RAG augmentation — all without any network access.

These results show that practical, intelligent assistants can be built entirely on-device using quantised models and efficient retrieval, without sacrificing the user experience to an unworkable degree. By eliminating cloud dependency, PocketLLM provides a genuinely private alternative to conventional AI assistants and offers a starting point for future research into scalable, efficient, and privacy-centric on-device intelligence.

Declarations

No human subjects were involved in this research. No personal user data was collected or analysed.

Data Availability: Source code and APK are available at https://pocketllmapp.vercel.app

Conflict of Interest: The authors declare no conflict of interest.

Funding: This research received no external funding.

References

1. T. Marques, S. Carreira, C. Grilo, and J. Ribeiro, "Revolutionizing Mobile Interaction: Enabling a 3 Billion Parameter GPT LLM on Mobile," arXiv preprint arXiv:2310.01434, 2023.
2. Y. Song, Y. Mi, H. Xie, and X. Jiang, "PowerInfer-2: Fast Large Language Model Inference on a Smartphone," arXiv preprint arXiv:2406.06282, 2024.
3. X. Zhao, Q. He, X. Chen, M. Xu, J. Ou, J. Yu, and J. Wan, "LLMCad: Fast and Scalable On-device Large Language Model Inference," arXiv preprint arXiv:2309.04255, 2023.
4. L. Li, R. Qin, W. Deng, L. Wen, Q. Su, Y. Wan, and M. Cheng, "Transformer-Lite: High-efficiency Deployment of Large Language Models on Mobile Phone GPUs," arXiv preprint arXiv:2403.20041, 2024.
5. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459–9474, 2020.
6. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. 2019 Conf. on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP), Hong

Kong, China, pp. 3982–3992, Nov. 2019.

7. sentence-transformers, "all-MiniLM-L6-v2," Hugging Face, 2021. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
8. arcee-ai, "Arcee-Lite: A Compact 1.5B Parameter Language Model," Hugging Face, 2024. [Online]. Available: https://huggingface.co/arcee-ai/arcee-lite
9. G. Gerganov, "llama.cpp: LLM Inference in C/C++," GitHub, 2023. [Online]. Available: https://github.com/ggerganov/llama.cpp
10. D. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," arXiv preprint arXiv:2210.17323, 2022.
11. T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," in Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
12. W. Kwon, Z. Li, S. Zhuang, Y. Sheng, G. Zheng, H. Zhang, J. E. Gonzalez, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in Proc. 29th ACM Symposium on Operating Systems Principles (SOSP), 2023.
13. E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in Proc. 10th International Conference on Learning Representations (ICLR), 2022.
14. S. Mehta and M. Hannan, "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer," arXiv preprint arXiv:2110.02178, 2021.
15. W. Yin, Z. Li, and M. Guo, "LLM as a System Service on Mobile Devices," arXiv preprint arXiv:2403.11805, 2024.

Figures

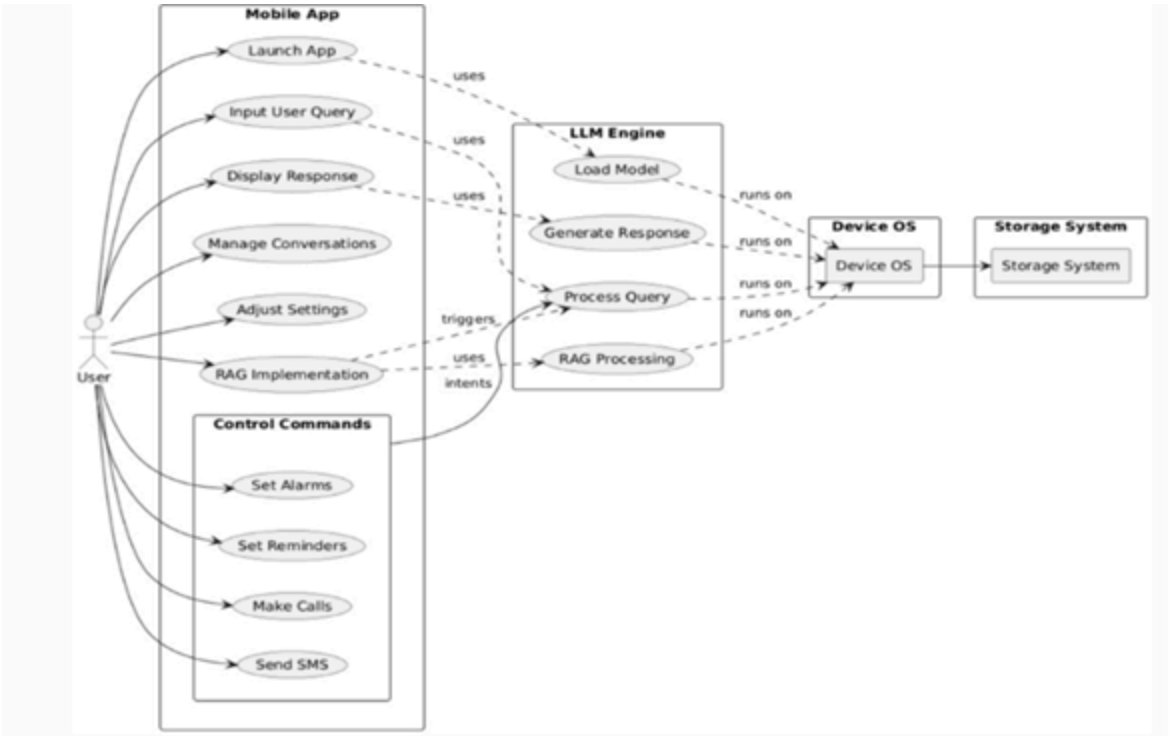


Figure 1

Figure 3.1: User Interface Diagram

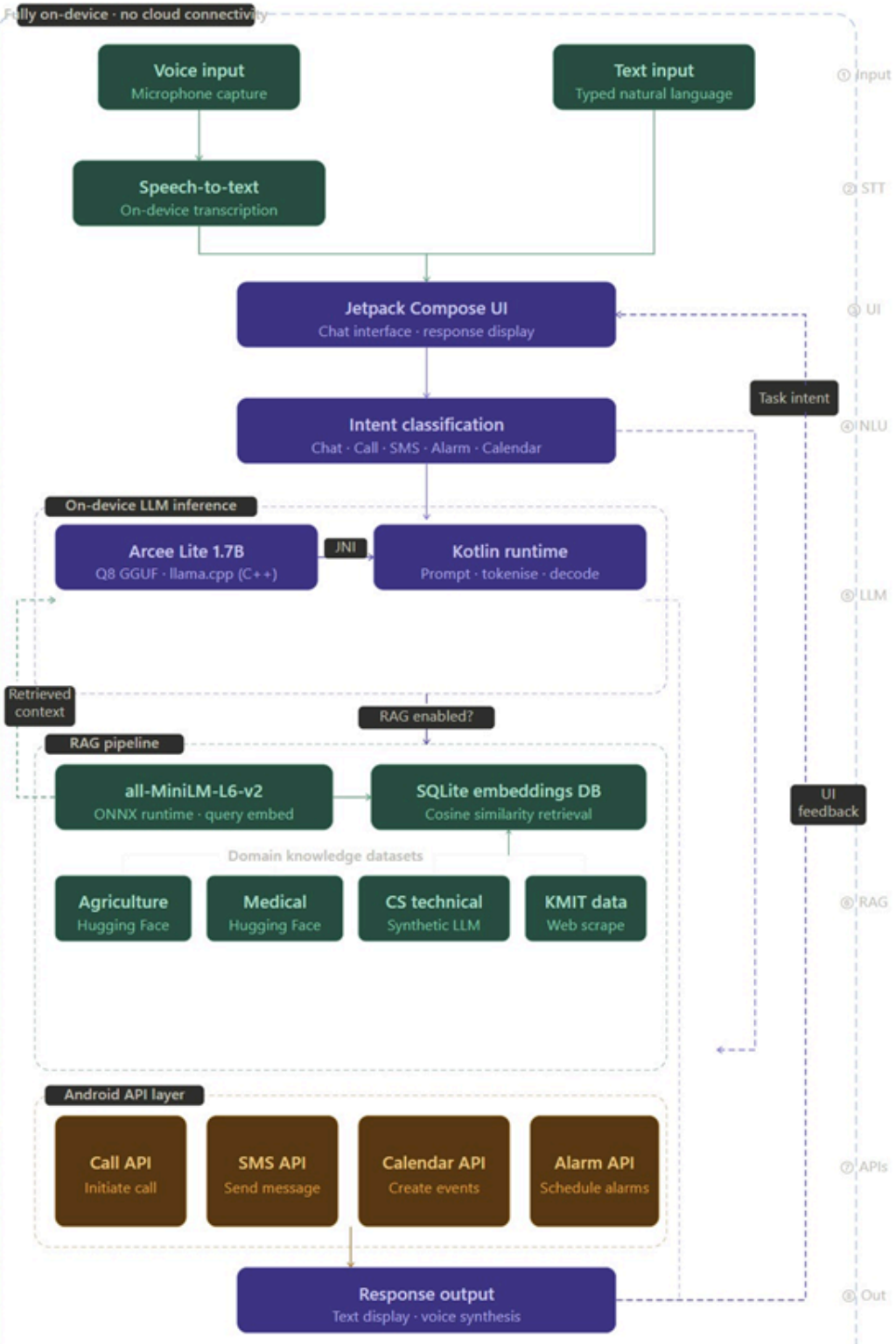


Figure 2

Figure 3.2: PocketLLM System Architecture Diagram

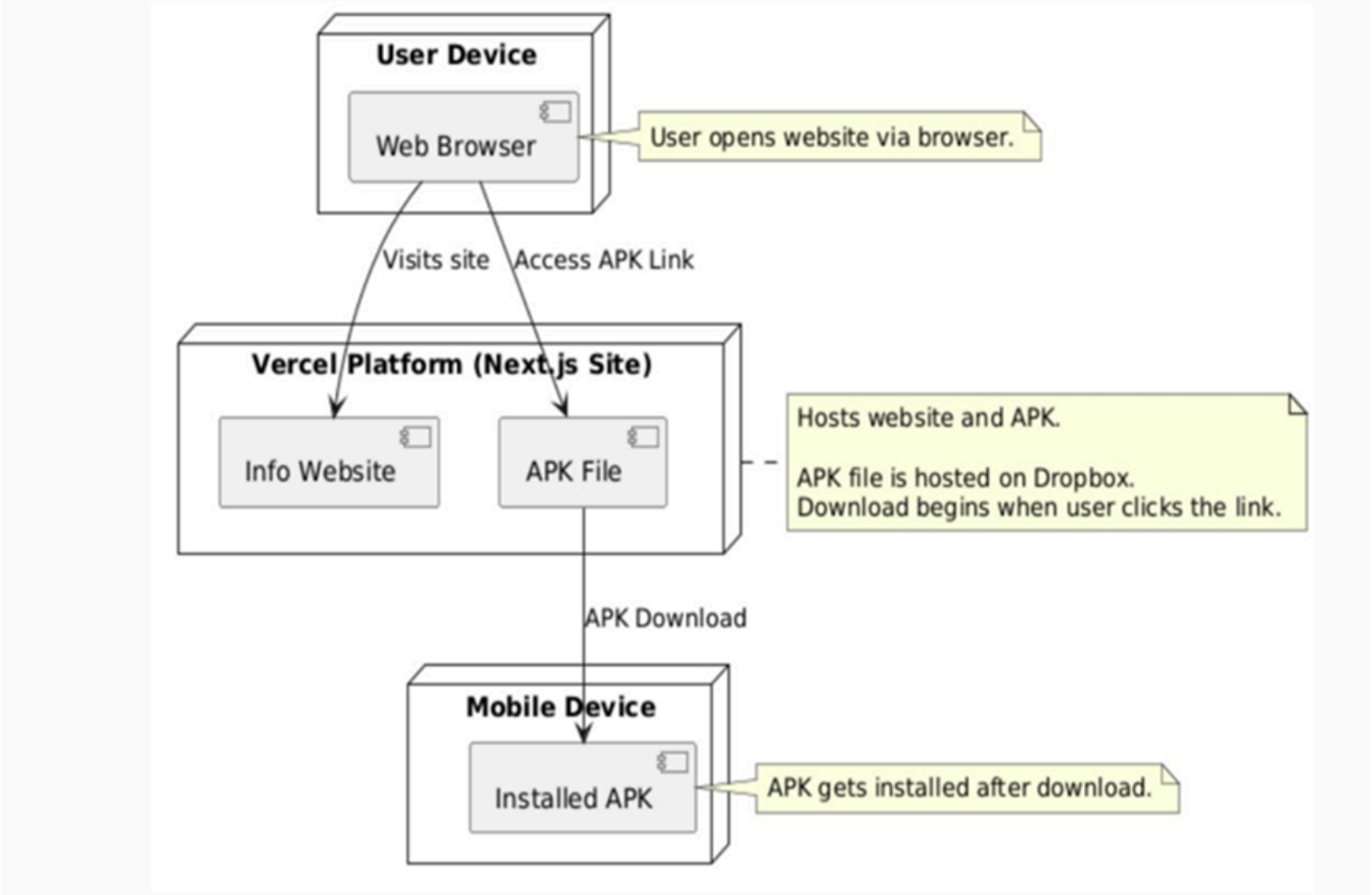


Figure 3

Figure 4.1: Deployment Diagram

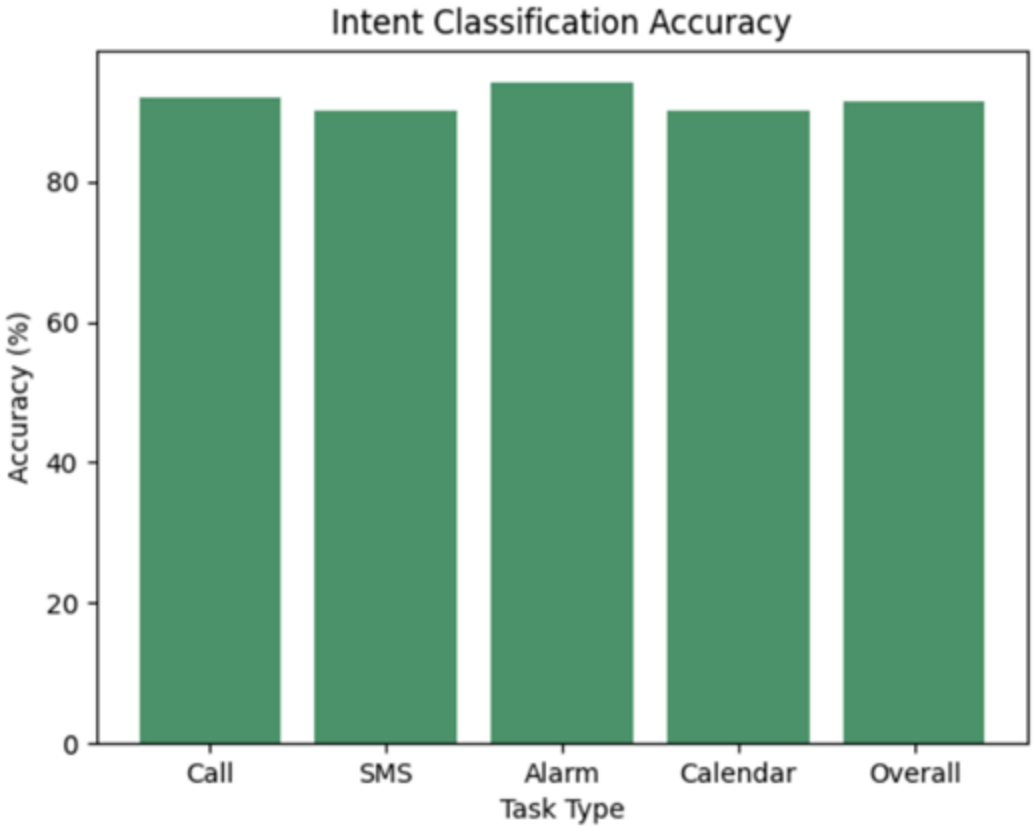


Figure 4

Figure 5.1: Intent Classification Accuracy Graph