AI-Powered Tutor Project: Final Documentation

Team BitByBit (Anushka Sinha, Pariza, Pranjal Garg, Kartik Gupta)

December 1, 2025

Contents

1 Project Overview & Journey

1.1 Introduction

The project aimed to develop an Android application that performs advanced conversational AI tasks, such as chat, quiz generation, and document analysis, entirely on the mobile device's dedicated hardware (the **Hexagon Tensor Processor/NPU**). This shifts the paradigm from cloud-based to **Edge AI**, emphasizing privacy, low latency, and power efficiency.

1.2 Development Timeline and Key Findings

The project followed a structured approach, starting with a basic proof-of-concept and iterating towards NPU-accelerated features. Prior to the final NPU implementation, the team utilized CPU-based inference models (GGUF/Ollama) to build core application logic.

Table 1: Project Development Timeline and Key Decisions

Date	Key Activity & Decision	Outcome & Finding
Aug 18, 2025	Initial meeting with Professor Karthik Vaidyanathan.	Decision: Focus on RAG chatbot using existing Qualcomm/Llama models.
Aug 25, 2025	Meeting with Qualcomm Team.	Recommendation: LLAMA model integration. Began initial implementation steps.
Sep 8, 2025	Implementation with Ollama/GGUF.	Llama operational on device using QIDK and running via **Ollama/GGUF**, utilizing the **CPU** for inference. Successfully integrated hardcoded PDF/image parsing.
Sep 15, 2025	Model Export Attempt and Pivot.	Primary Objective: Compile and export large models (**Llama 3.1 8B**, **Llama 2_7B_chat**) into **GGUF** format. Failed due to severe memory/processor constraints. Immediate shift to testing a smaller model: **Phi-3.5-mini-instruct**.

Continued on next page

Table 1 – continued from previous page

Date	Key Activity & Decision	Outcome & Finding
Pre-Oct 6, 2025	Implementation Status.	All core features (chat, parsing, etc.) were fully built and running on the **CPU** using the GGUF model framework, establishing functional correctness.
Oct 6, 2025	Critical Pivot to NPU.	Fixed PDF parsing. Started the critical effort to convert the entire application from **CPU** inference to **NPU** acceleration, as suggested by the Qualcomm Team.
Final	Successful Deployment.	Final deployment achieved by successfully integrating **Llama 3.2 3B Instruct (INT8)** accelerated by the **Hexagon Tensor Processor (HTP)** via the **Genie framework**.

2 Technical Stack and Architecture

2.1 Hardware and Frameworks

The core innovation lies in the use of specialized hardware and a proprietary framework for efficiency.

- **Hardware Acceleration**: Qualcomm Snapdragon Platform, specifically the **Hexagon Tensor Processor (HTP)**, which is the dedicated **Neural Processing Unit (NPU)**.
- **AI Framework**: **Qualcomm Genie** (Generative AI for Inference and Execution).
- **AI Runtime**: **Qualcomm AI Engine Direct (QNN) Runtime**.
- **Model**: **Llama 3.2 3B Instruct** (INT8 Quantized).

2.2 System Architecture (High-Level Codebase Documentation)

The application is structured in distinct layers, ensuring separation of concerns. Figure 1 illustrates the high-level architecture.

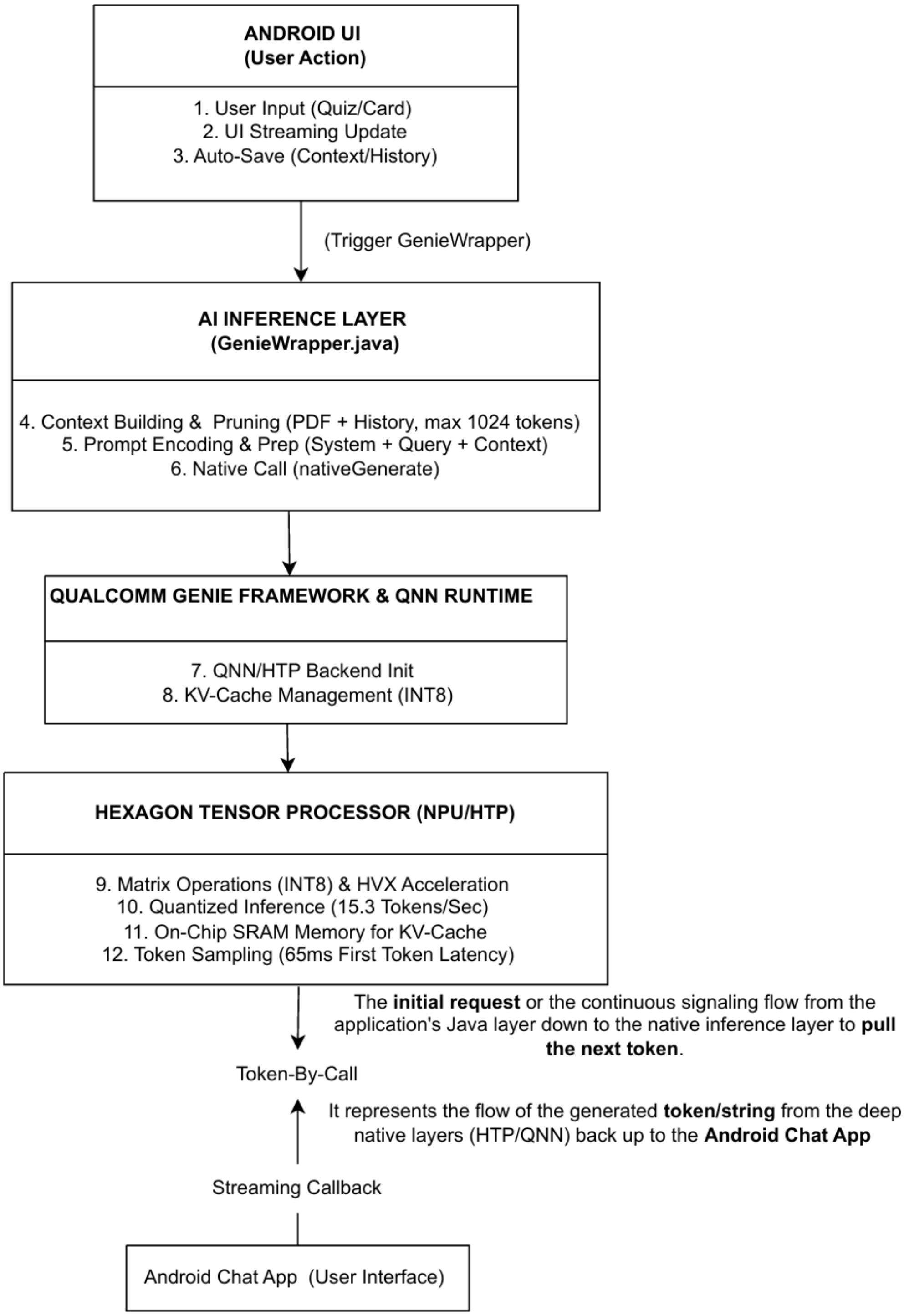

Figure 1: System Architecture of the AI-Powered Educational Tutor (Conceptual Diagram)

1. **Android UI Layer**: Handles user interaction and visual presentation.
    - **Key Files**: `Conversation.java` (Chat), `QuizActivity.java` (Quiz Interface), `HomeActivity.java`.
2. **Business Logic Layer**: Manages data, sessions, and non-AI processing.

    - **Key Files**: `ChatSessionManager.java` (using **Gson** for JSON serialization/persistence), `WeakTopicsAnalyzer.java` (analyzes quiz history), **PDFBox** library integration for document parsing.

3. **AI Inference Layer**: The bridge between Java/Android and the native Qualcomm AI frameworks.
    - **Key Files**: `GenieWrapper.java` (Java wrapper for native C++ functions).
    - **Core Logic**: Handles **Prompt Engineering**, **Context Management** (dynamic pruning to fit the 1024 token limit), and **Token Streaming**.

4. **Qualcomm NPU (HTP) Hardware Layer**: The on-device execution engine.
    - **Components**: QNN Runtime (`libQnnSystem.so`, `libQnnHtp.so`) and the compiled Llama 3.2 model binary.

2.3 Qualcomm Genie/NPU Integration (`GenieWrapper.java`)

The `GenieWrapper.java` class is the most critical component, housing the logic for LLM deployment and inference.

- **Model Quantization**: The original Llama 3.2 3B model (FP32, ∼12 GB) was quantized to **INT8** precision, resulting in a model size reduction to ∼1.8 GB (85% reduction).
- **Initialization**: The class loads the necessary native libraries and initializes the QNN runtime with the HTP backend, allocating the **KV-cache** in HTP memory.

```java
public class GenieWrapper {
    static {
        // Load QNN native libraries
        System.loadLibrary("QnnHtp");       // HTP backend
        System.loadLibrary("QnnSystem");    // Core runtime
        System.loadLibrary("genie_wrapper"); // Genie bridge
    }

    // Native method declarations
    private native long nativeInit(String modelDir, String configPath);
    // ...
}

// Initialization flow
genieWrapper = new GenieWrapper(
    "/data/local/tmp/genie_bundle",  // Model binaries location
    "genie_config.json"              // HTP configuration
);
```

Listing 1: GenieWrapper.java - Native Library Loading and Initialization

- **Inference Pipeline**: The `nativeGenerateStreaming()` method manages the core token generation flow. The **HTP** performs INT8 matrix multiplications, utilizing **Hexagon Vector Extensions (HVX)** for acceleration. A **Token-by-Token Generation** loop streams the output back to the Java UI layer.

3 Core Features and Codebase Implementation

3.1 Conversational AI Chat

The chat system is context-aware and designed for streaming responses.

- **Context Management**: The system dynamically constructs a context string, strictly adhering to the **1024 token limit**.
  - **PDF/Image Context**: Truncated to ∼625 tokens (2500 characters).
  - **Conversation History**: Sliding window of the last 6 messages (∼150 tokens).
- **Streaming UI**: Achieves an average performance of **15.3 Tokens/Second** on the HTP, providing a responsive experience.

3.2 Intelligent Quiz Generation

The LLM generates structured multiple-choice quizzes based on content.

- **Prompt Engineering**: Uses a structured `QUIZ_TEMPLATE` to guide the LLM's output format.
- **Personalization**: The `WeakTopicsAnalyzer.java` class manages this feature. It analyzes the student's past quiz results to identify topics with an accuracy **below 60%** and injects these weak topics into the quiz generation prompt.

```java
public class WeakTopicsAnalyzer {
    public static List<String> analyzeWeaknesses(String username) {
        // ... (Logic to load quiz history)
        Map<String, Double> topicAccuracy = new HashMap<>();

        // Analyze past performance and calculate accuracy per topic

        // Return topics with < 60% accuracy
        return topicAccuracy.entrySet().stream()
            .filter(e -> e.getValue() < 0.6)
            .map(Map.Entry::getKey)
            .collect(Collectors.toList());
    }
}
```

Listing 2: Weak Topics Analyzer (Personalization Logic)

3.3 PDF Processing Pipeline (`PDFProcessing.java`)

This pipeline is crucial for preparing raw document text for the NPU context.

- **Text Extraction**: Utilizes the **PDFBox** Android library.
- **Text Cleaning**: The `cleanPdfText` method removes excessive whitespace, fixes hyphenation issues, and removes common headers/footers (e.g., page numbers).

- **Truncation**: Text is strictly truncated to **2500 characters** (~625 tokens) to ensure the context fits the 1024-token budget, preventing the **"0 tokens" error**.

```java
private String cleanPdfText(String rawText) {
    // Remove excessive whitespace
    String cleaned = rawText.replaceAll("\\s+", " ");

    // Remove page numbers and headers
    cleaned = cleaned.replaceAll("(?m)^Page \\d+.*$", "");

    // Fix hyphenation from PDF line breaks
    cleaned = cleaned.replaceAll("-\\n([a-z])", "$1");

    // ... other cleaning steps

    // Truncate to fit token budget (2500 chars ~ 625 tokens)
    if (cleaned.length() > 2500) {
        cleaned = cleaned.substring(0, 2500);
    }

    return cleaned.trim();
}
```

Listing 3: PDF Text Cleaning and Truncation

4 Performance & Findings

The utilization of the HTP provided significant performance gains over standard mobile CPU/GPU inference.

Table 2: NPU vs. CPU/GPU Performance Benchmark (Llama 3.2 3B INT8)

Metric	NPU (HTP)	CPU (ARM)	GPU (Adreno)
Tokens/Second	15.3	2.5	8.2
First Token Latency	65ms	400ms	122ms
Power Consumption	1.1W	4.5W	3.2W
Inference Speedup	**6x** vs. CPU	1x	3.3x vs. CPU
Power Saving	**70%** vs. GPU	-	-

4.1 Key Lessons Learned

- **Context Management is Paramount**: The total context (PDF + History + System Prompt + User Query) must not exceed the 1024 token limit to avoid the critical **"0 tokens" error**. Dynamic pruning was the definitive solution.

- **Quantization is Necessary**: Using the **INT8 quantized Llama model** was essential for achieving the 85% size reduction, making the 1.8 GB model fit within mobile memory constraints.

- **Streaming Improves UX**: Streaming tokens significantly lowered **perceived latency** (first token in 65ms) and improved the user experience, despite longer total generation times.

5 Future Enhancements: Potential Focus Areas

Future development will target three key areas to enhance accessibility, personalization, and efficiency.

- **Voice I/O**
  Adding **Speech-to-Text** and **Text-to-Speech (TTS)** capabilities for accessible, hands-free tutoring.
- **LoRA Adapters**
  Implementing **Personalized Fine-Tuning** using LoRA (Low-Rank Adaptation) for specific subjects (e.g., Math, History). These specialized adapters will be loaded dynamically based on the student's current workspace or focus area.
- **KV Cache Optimization**
  Further tuning of the **Key-Value Cache** to allow for **larger PDF inputs** without degrading the tokens-per-second speed. This addresses the current context window limitation and improves document analysis capacity.

A Appendix: Technical Specifications

A.1 Software Versions

Table 3: Project Software Stack

Component	Version
Android SDK	34 (API Level 34)
MinSDK	31 (Android 12)
TargetSDK	34 (Android 14)
QNN Runtime	2.33.0.250327
Java	11
Gradle	8.11.1

A.2 Model Specifications

Table 4: LLM Model Specifications

Parameter	Value
Model Name	Llama 3.2 3B Instruct
Parameters	3.21 Billion
Quantization	INT8 (8-bit)
Model Size	1.8 GB
Context Length	1024 tokens