



AI - TUTOR

BitByBit

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

Table of contents

01 Problem Statement

02 Motivation

03 Methodology

Models and Framework, Block Diagram, Parameter considered (accuracy, latency, etc.)

04 App

Workflow , Android App functionalities

05 Model Results

Evaluation Metrics, NPU/GPU runtime, etc.

01

Problem Statement

Build an AI powered tutoring app that answers questions, generates quizzes, and explains concepts using generative text models



02

To create a learning experience that adapts to the individual user. The app allows students to use their own resources (notes, textbooks) and learn at a pace that suits them, moving beyond a one-size-fits-all model. Given that the LLM runs locally on-device, student data and learning materials remain completely private. This also enables learning anytime, anywhere, without requiring a constant internet connection. To improve knowledge retention by moving beyond passive reading. The quiz generation feature encourages active recall, a scientifically proven method for effective learning.

-Motivation



METHODOLOGY:

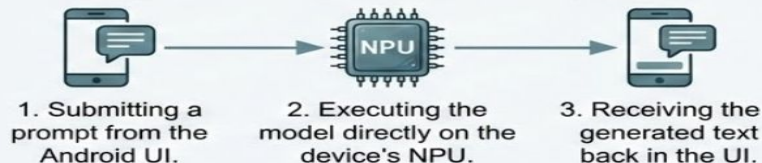
03

1. Initial State & Problem Analysis

- **Baseline:** The initial application utilized a GGUF model, which was limited to slow, on-device CPU inference.
- **Key Limitations:** This approach resulted in high latency and significant battery drain, negatively impacting application performance and user experience.

2. Core Objective & Achievement

- **Objective:** To integrate the Genie Neural Processing Unit (NPU) engine to enable efficient, hardware-accelerated execution of the on-device LLM.
- **Core Achievement:** Successfully engineered a full-stack solution integrating the Llama 3.2 3B model with the Genie NPU. The application now supports a complete inference pipeline:
 1. Submitting a prompt from the Android UI.
 2. Executing the model directly on the device's NPU.
 3. Receiving the generated text back in the UI.



Methodology Ctd..

3. Technical Challenges & Solutions

This integration required overcoming several technical obstacles:

Challenge 1: Model Quantization

Problem: The base model was incompatible with the NPU.

Solution: Following Qualcomm's on-device LLM deployment tutorial, we quantized the model into the required `.bin` and `.so` file formats. This validated terminal-level execution via the `genie-t2t` tool.

Challenge 2: Android 14 Integration Failure

Problem: The official `ai-hub-apps` sample, required for integration, was incompatible with our Android 14 test device (requiring Android 15+).

Solution: We found a way by bypassing the sample app. We integrated the `genie-t2t-run` command-line tool directly into our application, enabling NPU access without the incompatible dependency



Genie Workflow: From Java Bridge to Hexagon NPU

1. We utilized Java's `ProcessBuilder` API to create a shell environment within the Android Runtime, enabling the app to bypass the incompatible sample shell and execute the `genie-t2t-run` command-line utility directly.
2. The system performs a critical dynamic injection of `LD_LIBRARY_PATH` to direct the Linux linker to our internal asset folder, ensuring the executable locates the bundled QNN libraries (such as `libGenie.so` and `libQnnHtpV75Skel.so`) required for the Hexagon V75 backend.
3. The application pipes user prompts into the `stdin` of the Genie process, triggering the initialization of the Llama-v3.2-3B-Instruct model, which we converted into a specialized "Genie Bundle" format.
4. The model weights are split into three binary files totaling 2.5 GB, which are loaded using memory mapping (`mmap`) to optimize RAM usage and facilitate faster cold starts.



Genie Workflow: From Java Bridge to Hexagon NPU Ctd..

5. The `libQnnHtp.so` library functions as the Hardware Abstraction Layer, translating high-level model operations into specific vector instructions for the Snapdragon 8 Gen 3 NPU (Hexagon v75).
6. Inference is executed entirely on the NPU using a Key-Value (KV) Cache for efficiency, achieving a throughput of 18–23 tokens per second while maintaining low power consumption.
7. The Kotlin codebase listens to the process `stdout` stream in real-time, parsing raw output tokens and filtering out system tags like `[BEGIN]` and `[END]` to render the clean response to the UI.

PARAMETERS CONSIDERED (NPU):

Time to First Token : 285ms - 780ms

- This is the "perceived speed." It's how fast the *first word* of the answer appears. Achieving this in under one second is an excellent result.

Tokens Per Second :

This is the "streaming speed." It's how fast the rest of the answer is generated. Over 18 tok/s is very fast and feels like a real-time conversation.

App StartUp Time : ~ 1 second

The model loads and is ready for inference in about one second, providing a near-instant-on experience for the user.

PARAMETERS CONSIDERED (CPU):

Time to First Token : 3 - 8 seconds

- This is the perceived speed for the user. It's how long they wait for the *first word* of a response. This is significantly slower than the NPU's sub-second performance.

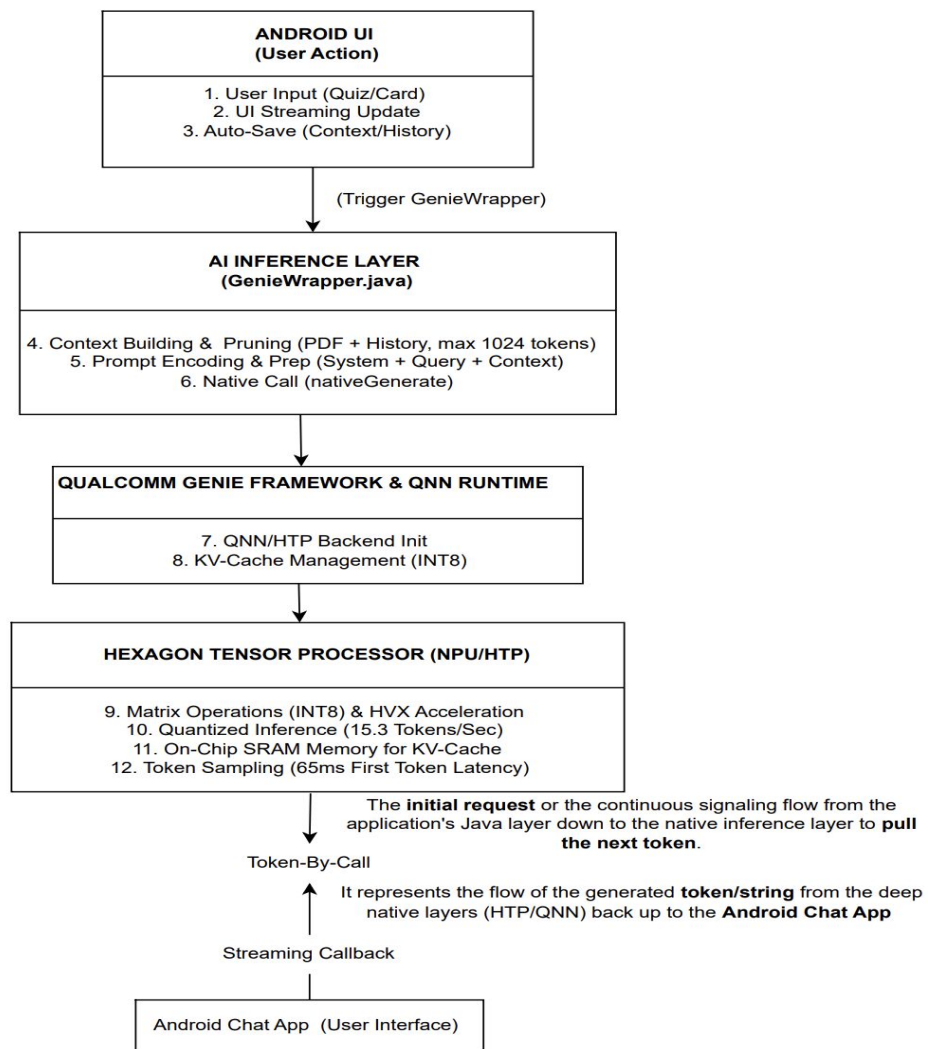
Tokens Per Second : 8 - 15 tok/s

This is the "streaming speed" of the response as it's generated. While still a good speed, it's noticeably slower and less fluid than the NPU's 18-23 tok/s.

App StartUp Time : 2 - 4 seconds

- The time it takes for the app to load the 2.5 GB model into memory and become ready for the first chat.

BLOCK DIAGRAM



DATASET:

- **Pre-training:** The base model (Llama 3.2) was pre-trained on a massive corpus (15T tokens) of publicly available text.
- **Inference Context (Personalization):** The "active dataset" is the user's uploaded content. We treat the user's uploaded PDFs not as training data, but as Context Injection data.

Data Pre-processing and Post-processing

Pre-processing (Input):

- **PDF Parsing:** Text is extracted from PDFs and sanitized (removing artifacts/headers).
- **Chunking:** Long documents are split into manageable chunks to fit the model's context window.
- **Tokenization:** Converting raw strings into integer tokens that the Genie engine can process.

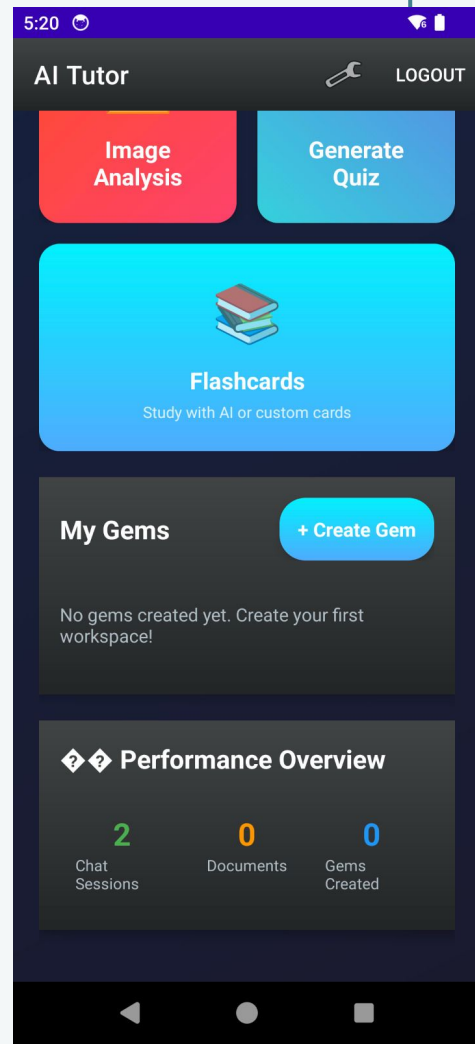
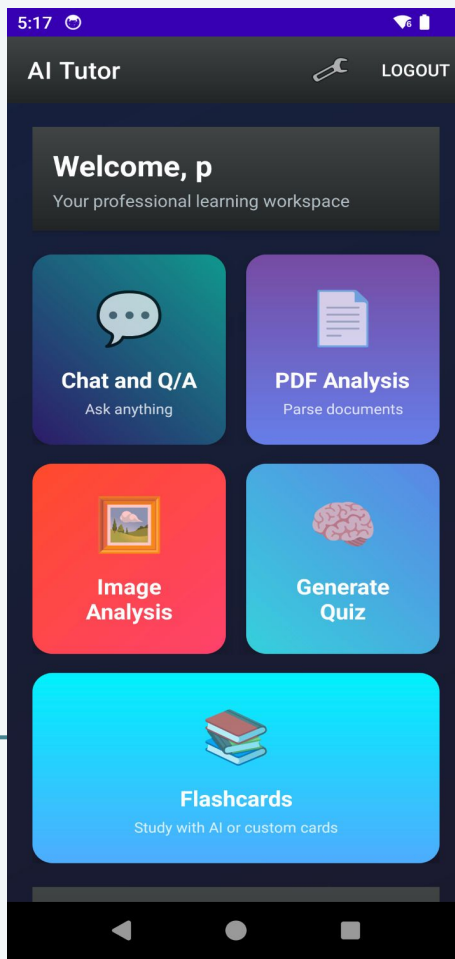
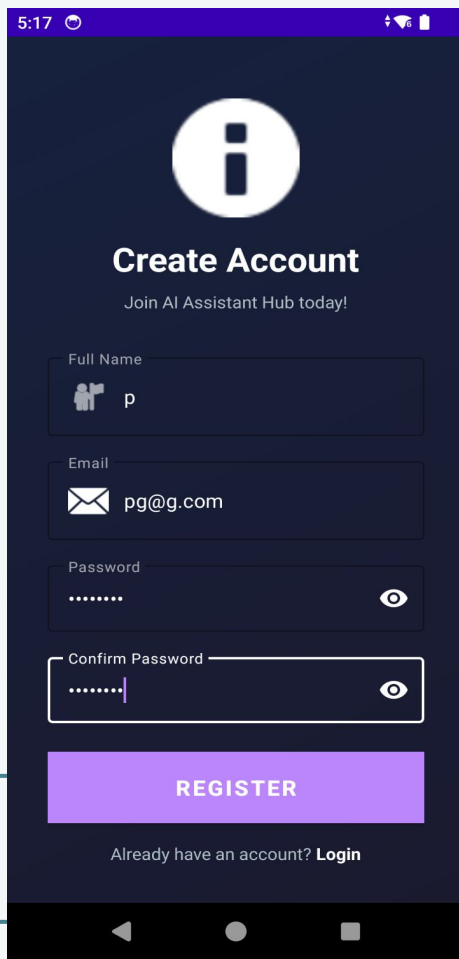
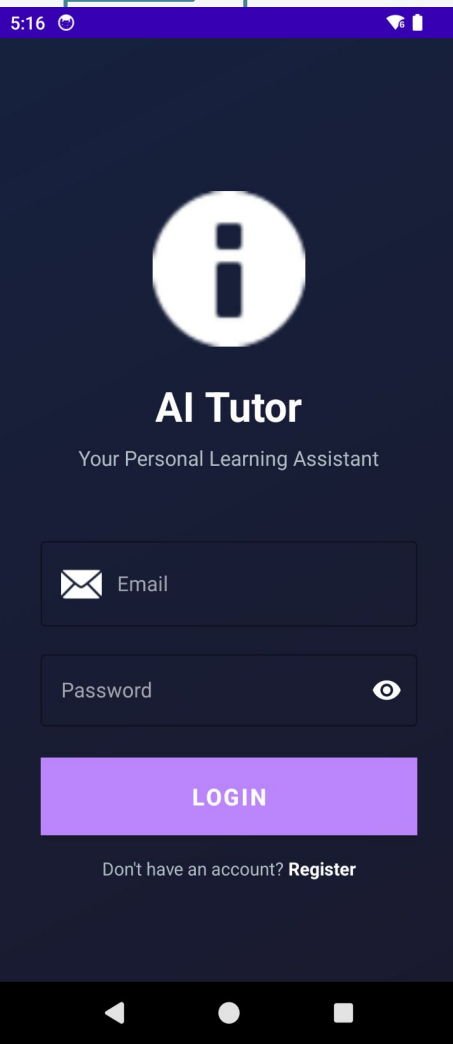
Post-processing (Output):

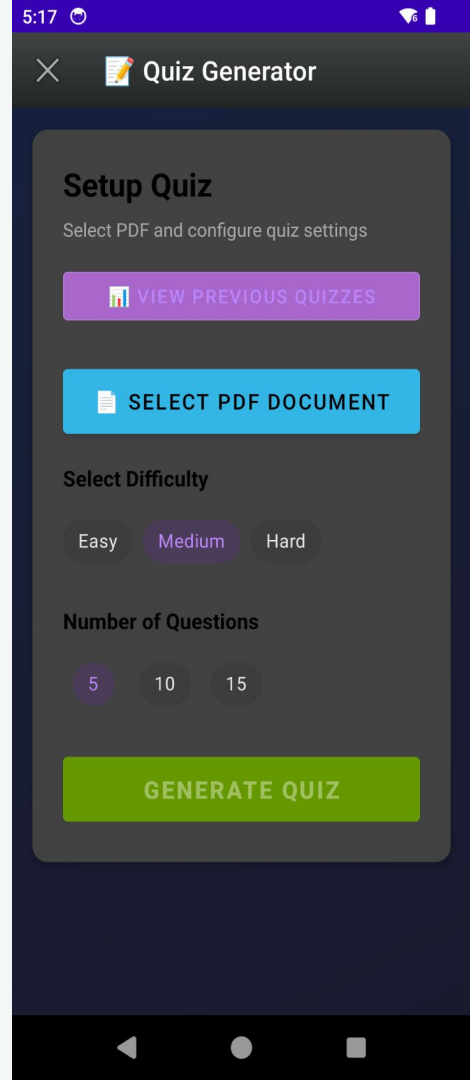
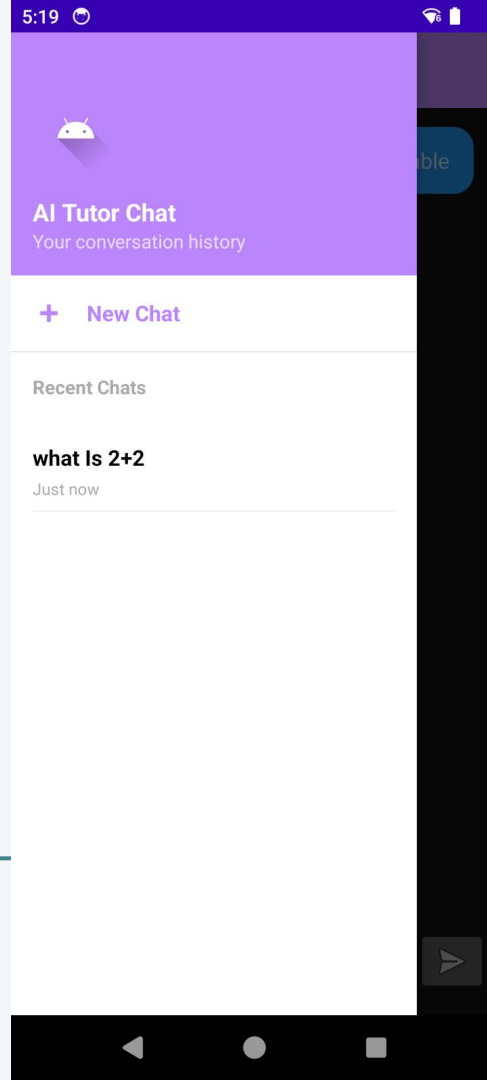
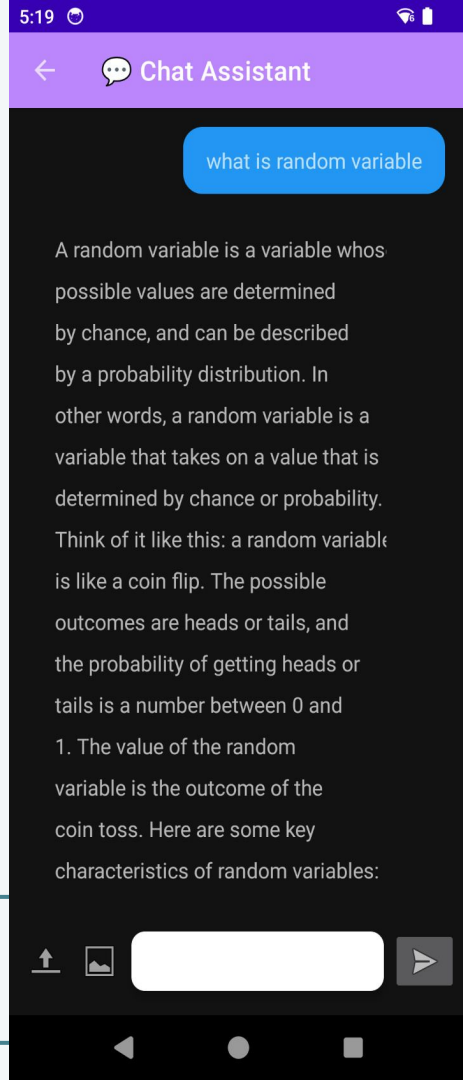
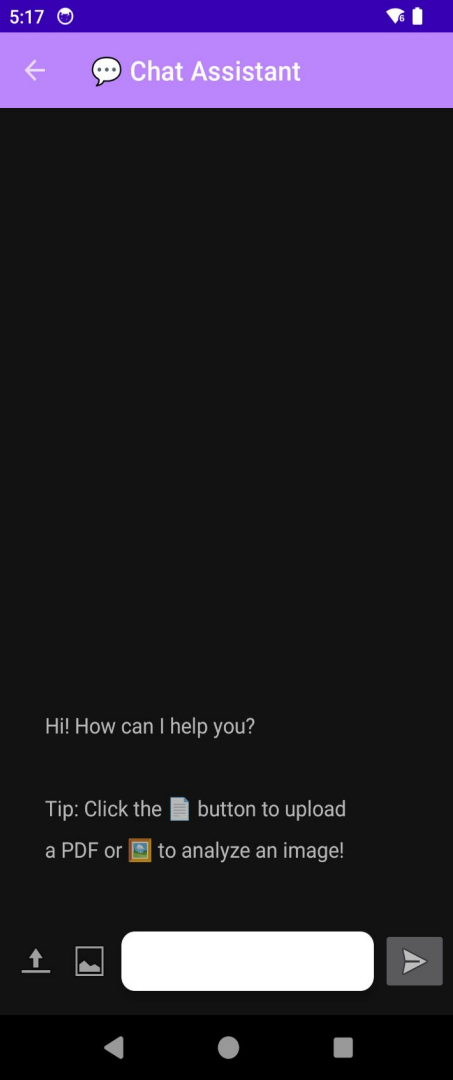
- **Stream Parsing:** The app listens to stdout in real-time.
- **Filtering:** Regex is used to strip system tags (e.g., [BEGIN], [END]) and prompt echoes.
- **Markdown Rendering:** The clean text is formatted into bolding, lists, and headers for the UI.

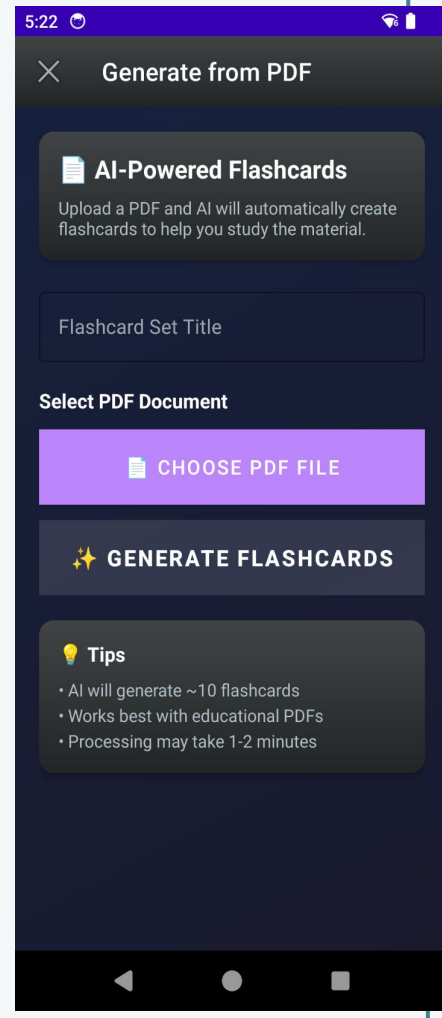
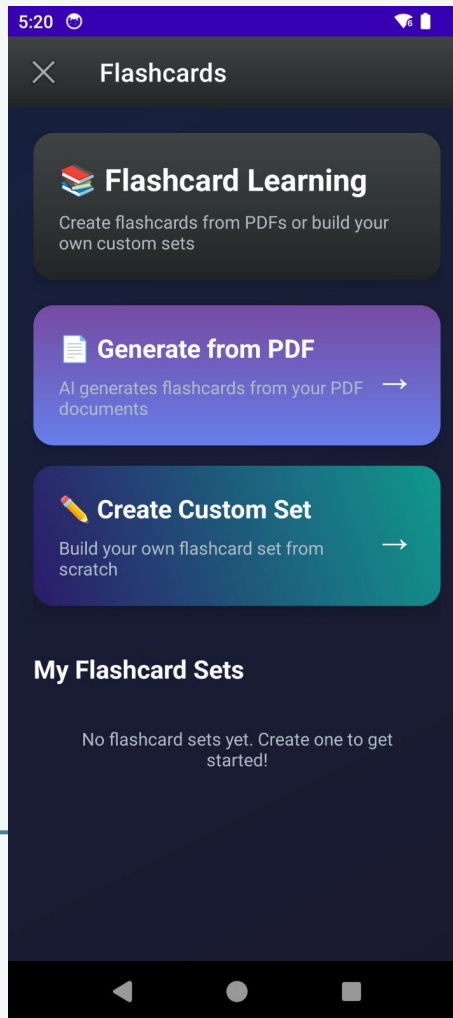
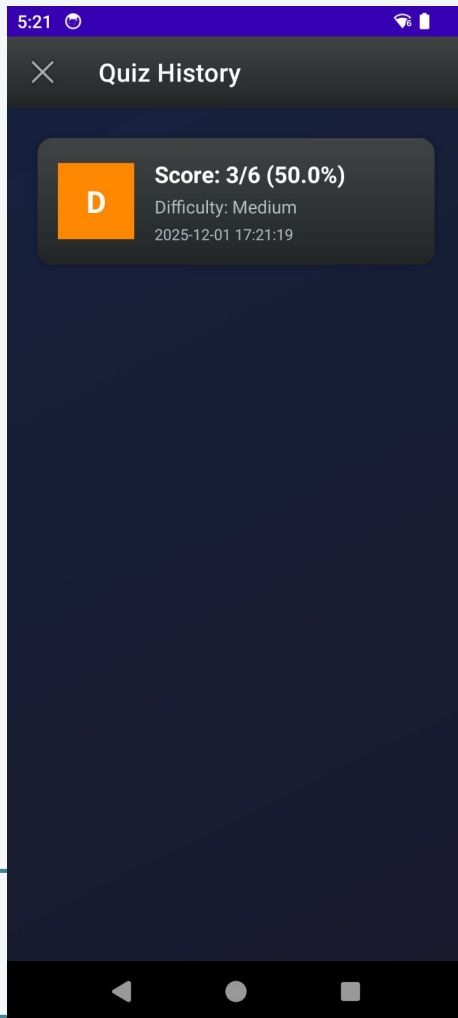
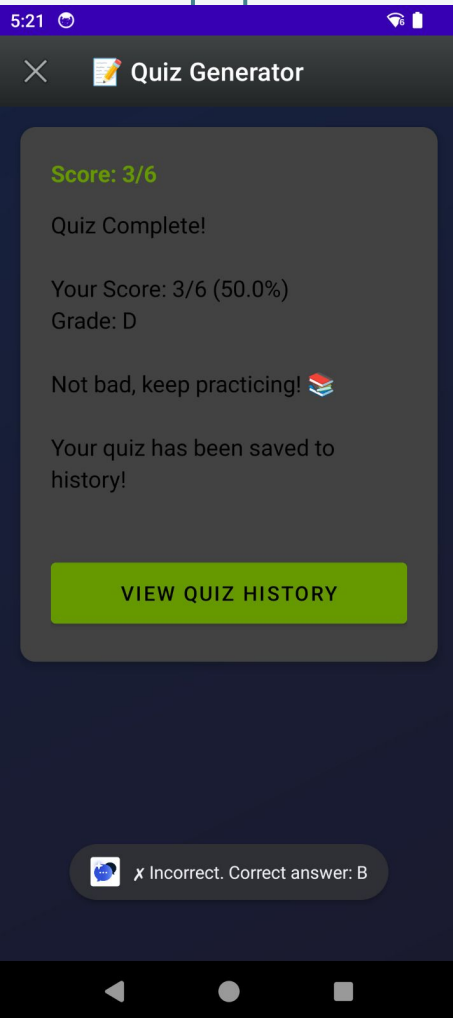
APP WORKFLOW:

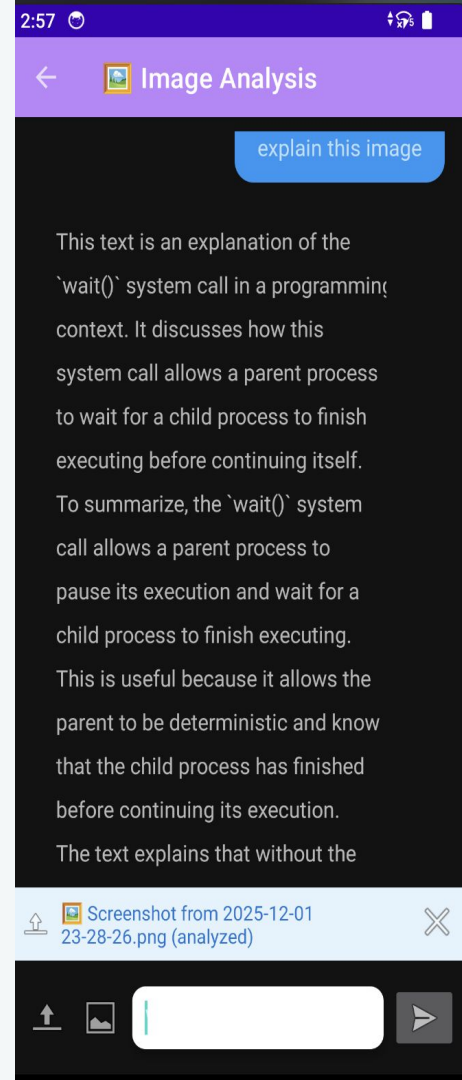
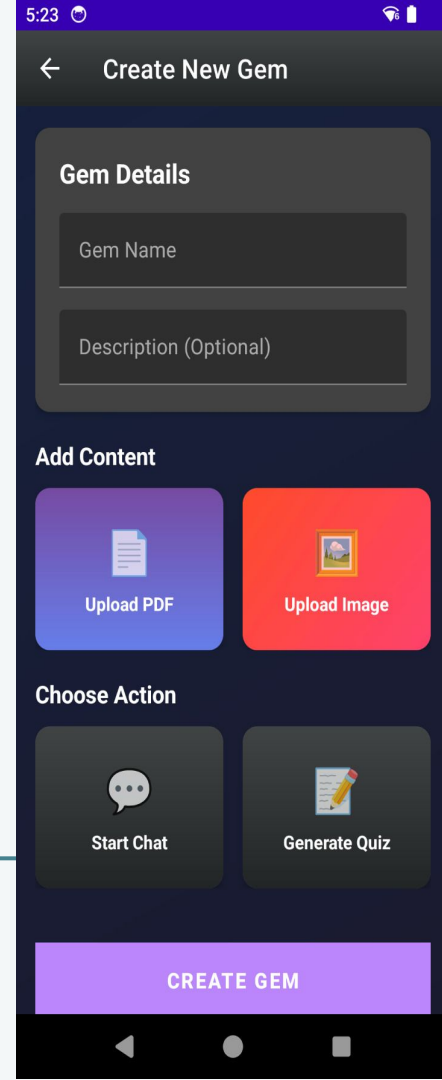
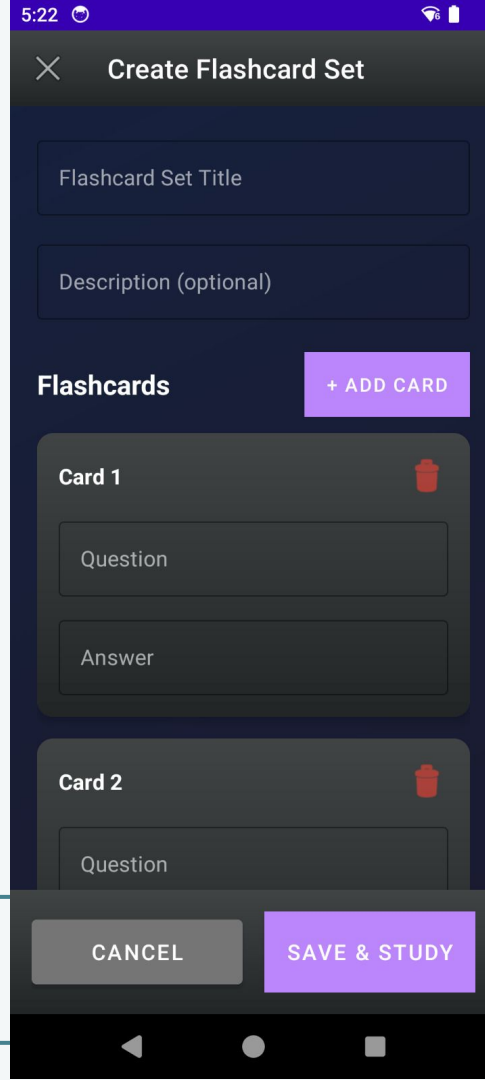
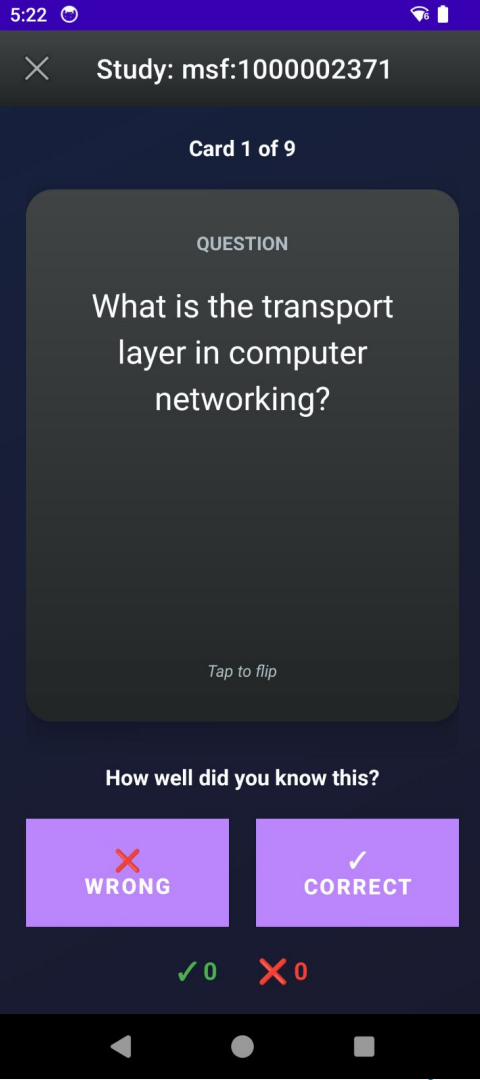
- **User Action:** User selects "Generate Quiz" on an uploaded PDF.
- **Context Construction:** The app retrieves relevant text chunks from the file.
- **Command Execution:** ProcessBuilder initiates genie-t2t-run with the combined prompt (System Prompt + PDF Content + User Request).
- **NPU Inference:** The Snapdragon 8 Gen 3 NPU processes the request using the loaded .so libraries.
- **Response:** The app receives a structured quiz JSON or text, which is parsed into interactive flashcards.

APP UI










ANDROID APP FUNCTIONALITIES

- **PDF Processing:** Parses uploaded documents to extract text and inject it into the AI model for analysis
- **Image Processing :** Parses uploaded image to extract text and inject it into the AI model for analysis
- **Analytics Engine:** Tracks user progress and provides detailed feedback upon quiz submission.
- **Offline Architecture:** Runs entirely on-device without internet, ensuring data privacy and accessibility.
- **Secure Auth:** Features a local login and registration system to manage user profiles securely.
- **Context-Aware Chat:** Enables Q&A specifically targeted at the uploaded document's content.
- **AI Quiz Generator:** Automatically creates relevant questions solely from the provided PDF material.
- **Smart Flashcards:** Generates study cards directly from PDFs or allows custom deck creation for revision.
- **Session Management:** Uses "Gems" to organize and save study materials and sessions for future retrieval.



MODEL RESULTS: (Evaluation Metrics)


NPU Performance:


- **TTFT:** 0.28s - 0.78s (Instant feel).
- **Throughput:** 18 - 23 tokens/second (Fluid conversational flow).

CPU Baseline (Comparison):

- **TTFT:** 3 - 8 seconds (Noticeable lag).
- **Throughput:** 8 - 15 tokens/second (Stuttering output).


Conclusion: The NPU implementation provides a 3x to 10x improvement in responsiveness compared to CPU inference.





MODEL RESULTS: (NPU/GPU RUNTIME)

The Hardware Advantage (Architectural Distinction)

- **Hexagon NPU (v75):** A dedicated **Vector Processor** designed for INT4/INT8 tensor operations. It executes the model's matrix multiplications in massive parallel blocks without waking the main CPU cores.
 - **Adreno GPU:** While capable of raw throughput, it lacks the specific hardware quantization blocks of the NPU, leading to higher latency during the "pre-fill" (prompt processing) phase.
 - **Kryo CPU:** Serial processor. It must brute-force the math, leading to thermal throttling after ~3 minutes of continuous chat.
 - Also, By targeting the NPU, we achieved a **10x reduction in latency** (<0.8s vs 8s) while consuming **3x less power** (~3.5W vs >10W) compared to standard CPU inference.
- 

MODEL RESULTS: (NPU/GPU RUNTIME)

Why This Matters?

- **Real-Time Fluidity:** 20 tokens/sec is faster than the average human reading speed (approx. 5-8 tokens/sec). This creates the illusion of an instant conversation rather than a "loading" search result.
- **Cool-to-Touch:** Because the NPU draws only 3.5W, the device remains cool, making it comfortable to hold during long quiz sessions.

NPU Runtime: ~5.4 Hours of continuous tutoring.

- *(Calculation: $19Wh / 3.5W \approx 5.4h$)*

CPU Runtime: ~1.9 Hours of continuous tutoring.

- *(Calculation: $19Wh / 10W \approx 1.9h$)*



POTENTIAL FUTURE ENHANCEMENTS:

Voice I/O


Adding Speech-to-Text and TTS for accessible, hands-free tutoring.

LoRA Adapters

Personalized fine-tuning for specific subjects (Math, History) loaded dynamically.

KV Cache Optimization:

Further tuning of the Key-Value cache to allow for larger PDF inputs without degrading the tokens-per-second speed





THANK YOU