SYSC 4907A – Progress Report

Group 76: AI Study Buddy

Due: December 5th, 2025

Team Members:

Haghighi Saed, Jaydon:	101206884
Chaudhury, Alvan:	101272922
Esenwa, Daniel:	101199099
Adotse-ogah, Marvel:	101259104

Supervisor:
Safaa Bedawi

Tables of Contents

1.0 Introduction

1.1 Project Overview

In this capstone project, we are building **AI Study Buddy**, a full-stack learning platform that combines an AI tutor with an objective focus-tracking system. The platform consists of three main technical components: a web application built with React and Vite, a Firebase backend (including Auth, Firestore, Storage, and Cloud Functions) integrated with **Genkit and Gemini** for conversational AI, and a **Raspberry Pi-based eye-focus tracker** that runs a deep learning model locally. Together, these components allow students to organize their studying into courses, sessions, and chats, interact with an AI study buddy that can explain concepts and guide problem-solving, and receive data-driven feedback on how consistently they stayed focused during each study session.

1.2 Problem Statement & Background

We are addressing the problem that many students struggle to maintain sustained focus and to manage their study time in a deliberate, data-informed way. Existing tools such as static planners, Pomodoro timers, and generic chatbots help to some extent with scheduling and answering questions, but they do not observe whether the student is actually paying attention to their work over time. At the same time, always-on camera monitoring raises significant privacy concerns, especially if video is streamed or stored in the cloud. This leaves a gap between purely self-reported productivity tools and invasive monitoring solutions. We see value in combining **AI tutoring** with **objective**, **privacy-preserving focus tracking**: the AI helps students understand material and plan their work, while the focus tracker provides an honest signal about engagement, enabling students to reflect not only on what they studied, but how they studied.

1.3 Proposed Solution

Our proposed solution is a tightly integrated system that connects an AI study assistant with an edge-based focus tracker and a cloud backend. On the web application, students authenticate, define courses and study sessions, and converse with the AI Study Buddy, which is backed by Firebase Cloud Functions, Genkit, and Gemini. The chat interface is organised around courses and sessions so that conversations remain contextually tied to what the student is studying. In parallel, a Raspberry Pi eye-focus tracker runs a TensorFlow Lite model on-device to classify whether the student is focused on the screen or distracted, applying temporal logic so that distraction is only flagged after a sustained period away from the screen. At the end of each study session, the Pi computes a **focus summary**, including metrics such as focus percentage, total focused versus distracted time, longest focused streak, number of distractions and uploads only this aggregated JSON to the backend. The web app then presents these summaries on a dashboard, allowing students to correlate their subjective study experience and AI interactions with objective focus data. This approach is novel in that it uses a **privacy-preserving edge device** to obtain meaningful behavioural signals, rather than streaming or storing raw video centrally, while still integrating tightly with a modern cloud and AI stack.

1.4 Project Scope and Goals

For this capstone, our scope is to deliver a working end-to-end prototype that demonstrates the feasibility and value of the combined system, rather than a production-ready product. In scope, we include: a functional web application with authentication, course/session/chat management, AI chat integrated with Gemini, basic file upload support, a first version of the focus summary dashboard, a Firebase backend with the necessary data models and Cloud Functions (including device pairing and summary ingestion), and a Raspberry Pi prototype capable of running a trained TensorFlow Lite focus model, applying our temporal focus logic, and uploading per-session summaries. We also aim to conduct an initial technical evaluation of the model's accuracy and the system's responsiveness. Explicitly out of scope for this phase are large-scale deployment and performance tuning, support for mobile or native clients, advanced analytics across many users, and long-term controlled user studies. We treat those as future extensions that build on the architecture and prototype we implement in this project.

2.0 The Engineering Project

2.1 Engineering Professionalism & Ethics

From the outset, we designed this project with privacy and ethical considerations at the centre, particularly because our system relies on camera input to estimate a student's focus. We treat video data as highly sensitive: all image processing and inference happen locally on the Raspberry Pi, and no raw video or identifiable frames are ever sent to the cloud. Instead, the Pi computes aggregated metrics such as total focused time, number of distractions, and focus percentage at the end of a session, and only this summary JSON is uploaded. This **summary-only** design significantly reduces the risk of misuse, unauthorized access, or re-identification from stored data.

We also recognise that students should not be monitored without their knowledge or consent. In a real deployment, we would require clear consent and provide a simple explanation of what is being measured, how the data is used, and what is not recorded (no video storage). Even at the prototype stage, we follow a similar mindset: we frame the system as a tool to help students reflect on their own habits, not as a surveillance mechanism. On the AI side, we aim for responsible behaviour from the Study Buddy by scoping it to academic support and avoiding prompts that encourage harmful or inappropriate content. We make it explicit that the AI can make mistakes and should not replace primary sources or instructors, and we design the system so that students are encouraged to verify critical information. Throughout our design, we adopt an informal compliance mindset: we minimize the data we collect, restrict what is stored long term to aggregated statistics, and ensure all communication between devices and the backend is authenticated and encrypted.

2.2 Project Management

We manage this capstone using an iterative, **Agile-inspired process**. Rather than planning one large waterfall sequence, we break the work into smaller milestones that each deliver a tangible increment: for example, "basic web chat without AI", "AI integration with Genkit and Gemini", "initial

Raspberry Pi capture and inference loop", and "end-to-end flow with summary upload and dashboard". At the beginning of each iteration, we define a small set of goals, distribute tasks based on team members' strengths, and track progress against those goals. This approach has helped us adapt to new insights, for instance, adjusting the focus model design after early experiments, without derailing the overall timeline.

In terms of tooling, we use Git and GitHub for version control and code reviews, ensuring that all significant changes go through peer review before being merged. We maintain a lightweight issue tracker on GitHub where we log tasks, bugs, and design decisions. For day-to-day communication, we rely on group chats and short stand-up style check-ins to synchronise on priorities and blockers. Our high-level timeline for the capstone is roughly: early weeks for requirements, architecture, and initial web/backend skeleton; mid-phase for AI integration, Raspberry Pi pipeline, and first working prototypes; and later weeks for refining the model, polishing the UI, testing, and preparing the final report and demonstration. This structure gives us clear checkpoints while leaving enough flexibility to respond to technical challenges.

2.3 Relation to Degree Program

This project is intentionally designed to bring together multiple strands of our degree program into a single, cohesive system. From a software engineering perspective, we apply principles of modular architecture, separation of concerns, and version control to design a system that cleanly separates frontend, backend, and edge device responsibilities. We also practice testing, incremental development, and documentation, which are key themes in our coursework. In machine learning, we draw on our understanding of deep learning, model training, data preprocessing, and evaluation to design and train the TensorFlow model that powers the eye-focus tracking on the Raspberry Pi.

On the **embedded and IoT side**, the Raspberry Pi component requires us to handle on-device deployment, performance constraints, and real-time processing of sensor data (the camera feed). This forces us to consider issues like model size, inference speed, and robust behaviour when the device boots, crashes, or loses connectivity. Finally, the backend leverages topics from databases and cloud computing: we use Firestore as a managed NoSQL store, Cloud Functions as a serverless compute layer, and Firebase Auth to secure access, effectively applying cloud-native design patterns from our coursework. By integrating all these aspects into one system, the project gives us a holistic, end-to-end experience of building a modern real-world application.

2.4 Individual and Team Contributions

Team Member	Primary Responsibilities
Jaydon	Frontend implementation (React + Vite), integrating Firebase on the client, building the chat UI, course/session views, and initial focus summary dashboard; integrating the Raspberry Pi subsystem with the backend APIs (pairing, currentSession, sessionSummary).
Marvel	Deep learning model design and experimentation in TensorFlow/Keras; working on data preprocessing pipelines and model architectures for focus vs distraction classification.
Danny	Data collection and labelling for the eye-focus model; preparing and cleaning datasets; setting up training runs and tracking results.
Alvan	Deployment of the trained model to TensorFlow Lite and Raspberry Pi; integrating the TFLite model into the Python agent and validating on-device performance (FPS, latency, stability).

2.4.1 Contributions to the Project

As a team, we divided responsibilities so that each member could lead in their area of strength while still collaborating across boundaries. One member has focused primarily on the frontend, implementing the React components for authentication, course/session/chat management, the chat interface, and the initial focus summary dashboard, as well as integrating the Firebase SDKs on the client side. As well as, the backend and cloud infrastructure, setting up Firebase Auth, Firestore data models, Storage rules, and Cloud Functions, and integrating Genkit and Gemini for AI chat sessions. Three members have concentrated on the machine learning and Raspberry Pi subsystem, including researching datasets, designing the TensorFlow model, setting up the camera and TensorFlow Lite runtime on the Pi, and implementing the focus state machine and summary generation. Throughout the project, we have also shared responsibilities for integration, debugging, and end-to-end testing, since many issues span multiple layers of the system.

2.4.2 Contributions to the Report

We approached the report in the same collaborative spirit. The team member leading the frontend has contributed most of the writing around the user experience, UI design, and client-side architecture, including descriptions of the React components and their interactions with Firebase. The backend-focused member has written the sections describing the overall system architecture, data models, Cloud Functions, and AI integration with Genkit and Gemini. The team member working on machine learning and the Raspberry Pi has authored the portions of the report that cover dataset selection, model design and training, TensorFlow Lite deployment, and on-device focus logic. We jointly refined the introduction, problem statement, and conclusion to ensure a consistent voice, and we reviewed each other's sections for technical accuracy and clarity. Diagrams, such as the high-level architecture and data flow between components, were prepared collaboratively using shared diagramming tools, with each member responsible for verifying the parts corresponding to their subsystem.

3.0 Objectives and Requirements

3.1 Objectives, Scope, and Success Criteria

Our primary objective in this capstone project is to build an AI-assisted study companion that helps students understand academic material while also giving them objective feedback on how consistently they stayed focused. Concretely, we want students to be able to organise their work into courses and sessions, chat with an AI study buddy that can answer questions and guide problem-solving, and then review a summary of their focus behaviour for each session, powered by an edge device rather than intrusive monitoring. A second core objective is to measure and quantify focus using a Raspberry Pi–based eye-focus tracker that runs a deep learning model locally, so that we can compute metrics like focus percentage, longest focused streak, and number of distractions without streaming or storing any raw video in the cloud.

For the scope of this capstone, our goal is to deliver a working end-to-end prototype that convincingly demonstrates this combined experience. At a minimum, we aim to implement: a web application where students can sign in, create courses/sessions/chats, and interact with an AI Study Buddy; a Firebase backend with the necessary data models and Cloud Functions; a TensorFlow-based focus model deployed on a Raspberry Pi using TensorFlow Lite; and the full path from local focus detection to summary upload and dashboard visualisation. We define success in terms of several measurable criteria: the system should support a complete demonstration workflow from login to focus summary; the AI chat should respond with acceptable latency and maintain session context; the focus model should achieve a reasonable accuracy on our validation data when distinguishing focused vs distracted states; the Pi should sustain an acceptable inference frame rate for our use case; and the entire pipeline (including summary upload and display) should behave reliably under normal conditions.

3.2 Functional Requirements

Functionally, the system must support a set of clear user-facing capabilities. From the student's perspective, they need to be able to create and manage an account, sign in securely, and organise their studying within the web application by creating courses, sessions, and chats. Within a selected chat, they should be able to interact with the AI Study Buddy, which uses context from previous messages and from the current course/session to provide more tailored responses. The interface must also allow a student to pair their Raspberry Pi device to their account using a simple, guided flow so that the Pi's focus summaries are correctly associated with their sessions. Finally, the student should be able to view focus summaries in the web app, including metrics such as total focused vs distracted time, focus percentage, longest focused streak, and number of distractions for a given study session.

Behind these user interactions, we have several device and backend functional requirements. On the Raspberry Pi, the focus tracker must be able to capture video locally, run the TensorFlow Lite model, and infer per-frame focus states, then apply our temporal logic (for example, only marking the student as distracted after about 30 seconds away from the screen). It must maintain a log of state transitions during a session and, when the session ends, compute and package a summary JSON with the relevant metrics. The Pi then needs to upload this session summary to the backend through an authenticated endpoint, using the device token it obtained during pairing. On the server side, our Firebase Cloud Functions must accept these summaries, store them in Firestore under the correct user and session, and make them available to the frontend through efficient queries. The web application must then retrieve and display these focus analytics to the student in a clear and interpretable way.

3.3 Non-Functional Requirements

In addition to functional behaviour, our system must satisfy several non-functional requirements. From a performance standpoint, the AI Study Buddy should respond within a reasonable time window so that conversation feels interactive; we target response latencies that are acceptable for a typical chat application, even when responses are streamed. On the Raspberry Pi, the TensorFlow Lite inference loop does not need high-end frame rates, but it should sustain a stable rate (for example, around 10–15 frames per second) so that our temporal logic (30 second distraction threshold, 1 second refocus threshold) is meaningful and robust.

We also care about reliability and availability. The web application and backend should be able to handle normal usage without crashes, and the Pi's focus agent should restart automatically on failure and resume normal operation when network connectivity returns. Security and privacy are central: all communication between the Pi, the web client, and the backend must be done over HTTPS; device tokens used by the Pi for authentication must be stored securely and validated on the server; and, most importantly, no raw video or image frames should ever be uploaded to or stored in the cloud. Only aggregated focus summaries are transmitted, and Firestore access should be restricted to authenticated users with the appropriate permissions. Finally, we require good usability: pairing a device should be as simple as entering a short claim code, the chat interface should be intuitive and responsive, and the focus dashboard should present metrics in a way that is easy for students to understand without needing a technical background.

3.4 Components and Facilities

To achieve these objectives, we rely on a set of hardware and software components, as well as some shared facilities. On the hardware side, the key element is a Raspberry Pi 5 equipped with 8 GB RAM, a Raspberry Pi AI HAT+, and a compatible camera module, which together provide the compute and sensing required for real-time, on-device focus estimation. On the user side, we assume access to a laptop or desktop PC with a modern web browser, which serves as the main interface for interacting with the AI Study Buddy and reviewing focus analytics.

On the software side, our frontend is built with React and Vite, using TypeScript for type safety. We use Firebase as our main backend platform: Firebase Authentication for user accounts and session management, Cloud Firestore as our primary database for courses, sessions, chats, messages, devices, and focus summaries, Firebase Storage for user-uploaded study materials, and Cloud Functions as our serverless backend, including integration with Genkit and Gemini for AI chat. For the deep learning pipeline, we use TensorFlow 2.x with Keras for training the focus model and TensorFlow Lite for deployment on the Raspberry Pi, together with Python and OpenCV for camera capture and preprocessing. As part of our development environment, we rely on standard tools such as Git/GitHub for version control, Node.js and npm for frontend tooling, and the Firebase CLI and Google Cloud infrastructure provided through our institution or personal accounts. Where appropriate, we also use lab resources such as shared development machines or test rigs for the Raspberry Pi, but the overall design is intentionally portable so that it can run on commodity hardware and cloud accounts after the capstone.

4.0 Research and Design

4.1 Research Findings

Before committing to a design, we surveyed prior work in three main areas: intelligent tutoring systems, focus and attention tracking, and edge computing for privacy-sensitive machine learning. Research on intelligent tutoring systems and AI study assistants showed that conversational agents can meaningfully support learning when they provide step-by-step explanations, encourage self-explanation, and adapt to the learner's context. However, we also noted that many systems focus purely on content and do not account for whether the learner is actually engaged with the material.

In parallel, we reviewed literature on eye-gaze and attention tracking, including gaze estimation, head pose, and screen-attention detection. Existing work demonstrates that gaze and head orientation are useful proxies for attentional state, but we also saw that frame-level predictions can be noisy, which motivated our decision to add a temporal state machine (e.g., only classifying a student as ***distracted*** after roughly 30 seconds away from the screen). We then looked at edge computing for privacy-sensitive ML, where recent studies emphasise processing sensitive sensor data locally and only transmitting derived features or aggregates. This strongly influenced our choice to process camera input entirely on the Raspberry Pi and to upload only session-level focus summaries. Overall, the key insights that shaped our design were:

1) conversational AI should be context-aware and responsive
2) attention estimation benefits from temporal smoothing rather than single-frame decisions
3) local processing is significantly better from a privacy and trust perspective than streaming raw video to the cloud.

4.2 Technology & Platform Comparisons

We evaluated several technology stacks before converging on our current choices. For the backend, we compared Firebase to alternatives such as Supabase and a custom Node.js + PostgreSQL setup. Supabase offers an attractive Postgres-based model and built-in auth, but we found Firebase better aligned with our need for realtime updates (via Firestore listeners), tight integration with serverless functions, and straightforward client SDKs. A fully custom Node/Postgres backend would have given us more control, especially for complex queries, but it would also have added significant operational overhead that we deemed out of scope for a capstone.

On the **frontend**, we considered React, Vue, and Svelte. Vue and Svelte have simpler mental models in some respects and can lead to smaller bundles, but our team has the strongest experience with React, and React has excellent ecosystem support, especially when combined with TypeScript. Given our timeline, we prioritised familiarity and ecosystem over experimenting with a new framework, and chose React with Vite for its fast development workflow.

For the **AI layer**, we compared using Gemini via Genkit to alternatives such as OpenAI or Anthropic APIs, as well as the idea of hosting a local LLM. External APIs like OpenAI and Anthropic provide strong models, but Genkit + Gemini integrates naturally with Google Cloud and Firebase, and Genkit's session abstraction reduces the amount of custom session handling code we need to write. Running a local LLM would improve data control but would require much more infrastructure and likely dedicated GPU resources, which is beyond the scope of this project.

For the **edge stack**, we considered TensorFlow/TensorFlow Lite versus PyTorch/ONNX, and Raspberry Pi versus alternative edge boards (e.g., Nvidia Jetson, Coral Dev Board). PyTorch plus ONNX Runtime is appealing for research, but TensorFlow Lite offers a very smooth path from Keras training to embedded inference. Since we train the model ourselves and deploy it on a Raspberry Pi, TensorFlow 2 + TFLite was a natural choice. Regarding hardware, Jetson-class devices offer more GPU compute, but they are more expensive and less common in educational settings. The Raspberry Pi 5 with an AI HAT provides enough compute for our lightweight model and is more accessible and familiar, which fits both our educational goals and realistic deployment scenarios.

4.3 High-Level System Architecture

At a high level, our system consists of three cooperating subsystems: the web application, the Firebase backend + AI services, and the Raspberry Pi edge device. The web app runs in the browser and interacts with Firebase Auth, Firestore, and Cloud Functions through the Firebase SDKs. It presents the chat interface, course/session management views, and the focus summary dashboard. The backend, implemented as Firebase Cloud Functions, exposes endpoints for:

- **AI chat** (`/chat`, `/createChat`, `/getSessionHistory`)
- **Device pairing** (`/device/register`, `/device/pairingStatus`)
- **Focus summary ingestion** (`/device/sessionSummary`).
- **Genkit and Gemini** are used inside these functions to manage AI sessions and generate responses.

In our final design, the Raspberry Pi acts purely as an authenticated client of the backend rather than exposing its own APIs. When booted, it enters a pairing mode where it registers its claim code and polls for pairing status until a user claims it via the web UI; once paired, it receives a device token and uses that for all subsequent communication. When a student starts or stops a study session in the web app, the frontend updates the backend (for example, by marking a given sessionId as active or inactive and associating it with a deviceId). The Pi periodically polls a lightweight Cloud Function endpoint `/device/currentSession` using its device token to discover whether there is an active session for that device and, if so, which sessionId it should attach to. On seeing a transition from ***no active session*** to ***active session***, the Pi starts its capture and focus-logging loop; on seeing the reverse transition, it stops logging, computes an aggregated summary, and posts it to the `/device/sessionSummary` endpoint. Firestore then stores these summaries, and the web app reads them back to render the focus dashboard. The main data flows in the system are: chat messages and AI responses between the web app, Cloud Functions, and Firestore; focus summaries from the Pi to the backend and then to the web dashboard; and device pairing and session-state messages between the Pi and backend, coordinated by user actions in the web app.

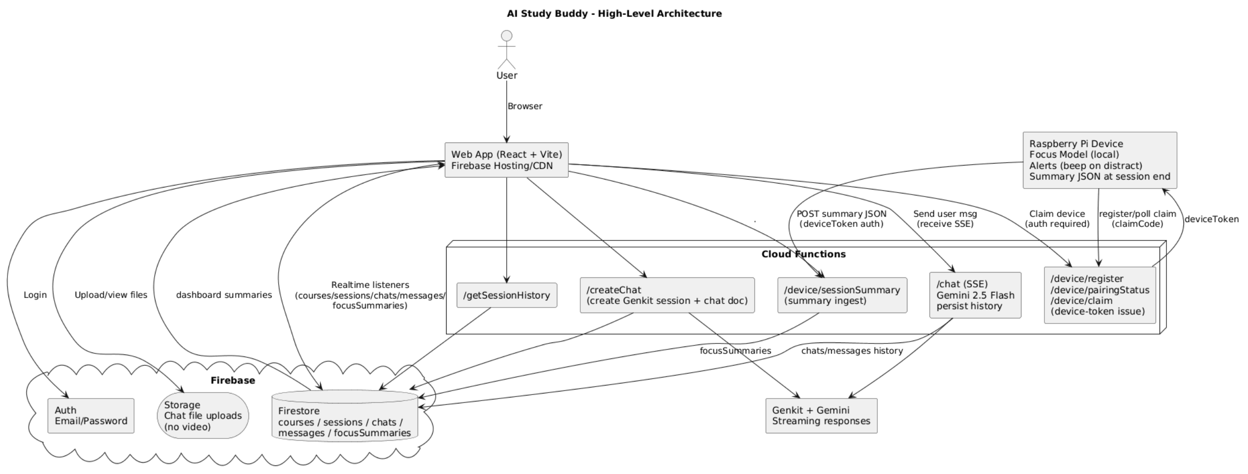

Figure 1: High-level Refined Block Diagram

4.4 Detailed Design Overview

4.4.1 Frontend (React + Vite + Firebase SDKs)

On the frontend, we structured the React application around a few core views: the authentication flow, the course/session/chat management view, the chat interface, and the focus summary dashboard. After authentication, the main layout allows a student to browse their courses and sessions in a sidebar, select or create chats within a session, and open a chat view where they can converse with the AI Study Buddy. This chat view is backed by Firestore listeners that keep the message list up to date and by service functions that call the Cloud Functions API for AI responses and chat creation.

The focus dashboard is designed as an additional view linked to a session. It subscribes to the focusSummaries data in Firestore for the current user and session, and presents metrics such as focus percentage, total focused vs distracted time, and longest streaks. We use the Firebase SDK to interact with Auth (for login state), Firestore (for realtime data), and Storage (for optional file uploads), keeping the client-side logic in small, focused hooks and components. Vite handles bundling and development tooling, giving us fast updates as we iterate on the UI.

4.4.2 Backend (Cloud Functions, Genkit, Gemini, Firestore)

The backend is implemented as a set of Firebase Cloud Functions that encapsulate our server-side logic. For the AI Study Buddy, we provide three main endpoints:

- /createChat: which creates a new Genkit session and a corresponding chats document in Firestore
- /chat: which loads the Genkit session, streams a Gemini response back to the client (via Server-Sent Events), and updates the session history in Firestore
- /getSessionHistory: which allows inspection of stored AI session state for debugging or analytics. Genkit manages the logical AI sessions, while we store the conversation history and state in a genkit_sessions collection using a custom session store.

For the device side, we define endpoints to support pairing and summary ingestion:

- /device/register: allows a Raspberry Pi to register its claim code and mark itself as awaiting claim.
- /device/pairingStatus: lets the Pi poll for pairing completion and retrieve a device token once a user has claimed it in the web UI.
- /device/sessionSummary: accepts a JSON summary from a Pi authenticated with a device token, validates and normalises the data, and writes it into Firestore under a **focusSummaries** structure keyed by user and session. All these functions enforce authentication and basic validation, ensuring that only authorised devices and users can register, claim, or write summaries.

4.4.3 Edge Device (Raspberry Pi + TensorFlow Lite)

On the Raspberry Pi, we implemented a dedicated Python agent that runs as a systemd service. This agent is responsible for capturing frames from the camera using OpenCV, preprocessing them (cropping and resizing to the input shape required by the model), and running inference using the TensorFlow Lite runtime. The TFLite model is a lightweight classifier that outputs a probability or label indicating whether the student appears focused on the screen or not. The Pi does not run a local HTTP server or accept inbound connections; instead, the agent operates as a long-running background process that makes outbound HTTPS requests to our Cloud Functions using a device token obtained during pairing.

We then layer a temporal state machine on top of the per-frame predictions. The agent keeps track of the current focus state and how long the model has been consistently predicting focused or not focused. If the model reports not focused continuously for roughly 30 seconds, the agent transitions to the DISTRACTED state; if it reports focused for approximately 3 second while in DISTRACTED, it transitions back to FOCUSED. Each state change is logged with a timestamp. During idle periods, the agent periodically calls a ***current session*** endpoint with its device token to determine whether there is an active study session associated with this device. When the backend indicates that a session has started, the agent begins camera capture and TensorFlow Lite inference, and starts accumulating an event log for that sessionId. When the backend later reports that the session has ended, the agent stops processing, computes a summary that includes total focused and distracted time, longest streaks, number of distractions, average focus before distraction, and overall focus percentage, writes this summary to a local JSON file, and then posts it to our /device/sessionSummary endpoint using the device token. This design avoids the complexity and security risks of exposing a server on the Pi, while still allowing the device to react promptly to user actions initiated from the web frontend.

4.4.4 Data Models & APIs

We designed our Firestore data model to mirror the main concepts in the application. The core collections are:

- **courses** (containing course metadata tied to a user)
- **sessions** (linked to courses and representing study sessions)
- **chats** (linked to sessions and representing AI chat threads)
- **messages** (linked to chats and storing the actual conversation content).

For the AI side, we maintain a **genkit_sessions** collection where Genkit session state and history are persisted. For device management, we have devices or a similar structure to record device IDs, claim codes, and their association with users.

For focus analytics, we maintain a **focusSummaries** structure keyed by user and session. Each summary document contains fields such as: (sessionId, deviceId, startTs, endTs, focusedMs, distractedMs, longestFocusedMs, longestDistractedMs, distractions, avgFocusBeforeDistractMs, and focusPercent). Our /device/sessionSummary API expects a JSON payload that matches or can be mapped to this schema. Other APIs include /chat, /createChat, and /getSessionHistory, which accept and return JSON payloads describing chat messages, chat metadata, and AI session state. By keeping our data models explicit and our API payloads structured, we make it easier to reason about the system, evolve the schema over time, and connect the frontend, backend, and edge device in a consistent way.

5.0 Project Progress

5.1 Methodology

Our development methodology for this project is intentionally iterative. Rather than trying to design and build the entire system in one pass, we work in short build–test cycles for each major subsystem: the web application, the backend/AI services, and the Raspberry Pi focus tracker. For the web and backend, we typically start with a minimal vertical slice, for example, a simple chat flow with static responses, then progressively replace mock behaviour with real Firebase, Genkit, and Gemini integrations. On the machine learning side, we follow a similar pattern: begin with a very simple baseline (or even a stub that outputs synthetic focus labels) and then replace it with a trained TensorFlow model once the data pipeline and deployment path are validated.

To coordinate between sub-teams, we loosely align around features rather than layers. For example, an "AI chat MVP" iteration involves frontend (chat UI), backend (Cloud Functions and Genkit wiring), and some data modelling in Firestore. A **focus summary MVP** iteration involves device-side logging on the Pi, backend summary ingestion, and a basic dashboard view. We maintain a shared task list where each feature is broken into frontend, backend, ML, and device subtasks, and we regroup regularly to resolve integration issues. For the ML workflow specifically, we follow a pipeline of **data collection → preprocessing → training → evaluation → export to TensorFlow Lite → on-device testing.** We treat each step as a check-point: we do not invest heavily in tuning until we know the model can be exported and run at acceptable speed on the Pi, and we do not optimize dashboards until the summary schema is stable. This staged, iterative methodology has helped us de-risk the project progressively.

5.2 Current Progress

At this stage of the capstone, we have implemented a significant portion of the web application. Students can create an account, sign in via Firebase Authentication, and access a main interface that supports courses, sessions, and chats. The chat view is wired to Firestore and Cloud Functions so that messages are stored in the database and AI responses are retrieved via our /chat endpoint using Genkit and Gemini. We have basic file upload support integrated with Firebase Storage, allowing users to attach study materials to their sessions. The UI is not yet fully polished, but it is functional enough to support an end-to-end study flow from login to AI conversation.

On the **backend**, we have set up the Firebase project, defined core Firestore collections (courses, sessions, chats, messages), and implemented the main Cloud Functions for AI chat and chat creation. Genkit is integrated with Gemini, and we are able to create and persist AI sessions, handle streaming chat responses, and save history in Firestore. We have also implemented early versions of the device-related endpoints, though some are currently using stub logic until the pairing and summary formats are finalised.

For the **Raspberry Pi**, we have a prototype Python agent running as a service that can initialise the camera, capture frames with OpenCV, and pass them through a simple inference stub. At the moment, this stub can either simulate focus/distracted outputs or run an initial TensorFlow Lite model for basic testing. We have verified that the Pi can sustain camera capture and inference at a modest frame rate and that we can log state changes over time. In combination, these pieces already allow us to demonstrate an early end-to-end flow: a student can chat with the AI Study Buddy on the web, the Pi can run its prototype focus logic during a session, and some form of summary can be generated and transmitted to the backend, although the exact metrics and UI are still evolving.

5.3 Ongoing Work

Our current focus is on turning the prototypes into more realistic and robust components. On the machine learning side, we are working on training the first real focus model using TensorFlow. This involves finalising our labelled dataset, experimenting with lightweight architectures that can run efficiently on the Raspberry Pi, and evaluating the model's performance on held-out data (validation and testing sets). Once we are satisfied with these results, we will export the model to TensorFlow Lite and integrate it into the Pi agent in place of the stub.

In parallel, we are implementing the full pairing flow between the Raspberry Pi and the web platform. This includes generating and managing claim codes on the backend, building the web UI for users to enter a code and claim a device, and updating the Pi agent to register, poll for pairing status, and store the resulting device token. At the frontend level, we are also building out the focus summary dashboard UI, which will subscribe to focusSummaries in Firestore and present session-level metrics in a clear visual format (for example, cards or charts showing focus percentage, total focused time, and longest streaks). As these features converge, we are incrementally testing the full study session experience from start to finish.

5.4 Next Steps and Planned Milestones

In the short term, our milestones are largely technical and integration-focused. We plan to complete the TensorFlow Lite integration on the Raspberry Pi, replacing the inference stub with the trained model and validating that we can maintain an acceptable frame rate and stable behaviour for the 30 second distraction and 1 second refocus thresholds. We also aim to finalise the device pairing and summary upload paths, ensuring that the Pi can obtain a device token, use it for authenticated communication, and reliably post summaries to the /device/sessionSummary endpoint even after reboots or temporary network outages. Once those pieces are stable, we will wire the web dashboard fully to Firestore so that focus summaries appear automatically for the correct user and session.In the medium term, our milestones shift more towards testing, refinement, and user experience. We plan to conduct structured testing of the focus model (both offline and on-device), gather informal user feedback on the UI (e.g., is the dashboard understandable, is the pairing flow clear), and improve error handling and edge cases (such as device failures or function timeouts). We will also spend time polishing the UI, tightening up copy and visual design so that the platform feels coherent and professional for the final demonstration. Finally, we will integrate these results into the report and prepare the materials necessary for our capstone presentation, including a stable demo script that shows the entire system in action.

5.5 Challenges and Lessons Learned

So far, we have encountered several technical and project-management challenges. On the data side, collecting and labelling examples of **focused vs distracted** behaviour has proven more involved than we initially expected. We needed to define clear labelling guidelines, ensure we had a variety of lighting conditions and camera angles, and handle borderline cases where it is not obvious whether the student is genuinely focused. This has reinforced the importance of having a well-specified problem and a disciplined approach to dataset creation in any ML-driven system.

Another challenge has been performance on the Raspberry Pi. Even with a relatively powerful Pi 5 and an AI HAT, we must carefully balance model complexity, input resolution, and frame rate to achieve smooth real-time behaviour. Although the Pi 5 with the AI HAT is very capable from experience we knew that the power was limited when it came to running deep learning models and it was clear that we needed to prioritise lightweight architectures and consider quantisation. This pushed us to be more pragmatic in our model design and to think of inference constraints early, not as an afterthought.

Designing good focus metrics has also required iteration. It is easy to compute simple aggregates like total focused time, but we found that metrics such as longest focused streak, number of distractions, and average focus before distraction provide a more nuanced picture of a study session. We have had to think carefully about which metrics are both meaningful and easy for students to interpret.

From a project-management perspective, we have learned the value of early integration. By wiring up basic end-to-end flows early, even with stubbed models or simplified UIs, we caught interface mismatches and design gaps that would have been much harder to fix later. We also learned that clear ownership (who is responsible for which subsystem) combined with regular cross-team check-ins is essential when working on a multi-component system. Overall, these challenges have helped us refine both our technical design and our approach to planning and coordination.

6.0 Testing and Evaluation

6.1 Testing Performed to Date

So far, our testing has been focused on verifying that each major subsystem behaves correctly on its own and that the most important end-to-end paths work in practice. On the frontend, we have primarily relied on structured manual testing: we exercise the authentication flow, creation of courses/sessions/chats, message sending, and file uploads, and we check that the UI reacts correctly to success and failure cases (for example, invalid credentials or network errors). We also validate that Firestore-backed views update in real time when data changes, and that navigation between views does not leave the app in inconsistent states.

On the backend, we have tested our Cloud Functions both interactively and via small API test scripts. For the AI endpoints (/createChat, /chat, /getSessionHistory), we verify that they enforce authentication, that they write and read the expected Firestore documents, and that they handle malformed input gracefully. We also test the Genkit integration by checking that sessions are created, streamed responses are delivered to the client, and histories are persisted correctly. Early versions of the device endpoints have been exercised using mock requests to ensure that claim codes and summary payloads are validated and stored correctly.

For the machine learning component, our testing so far has focused on training small prototype models and evaluating them on held-out validation sets. We monitor accuracy and, where appropriate, confusion matrices for the binary “focused vs distracted” classification problem to understand which types of errors are most common. These early experiments are not yet final, but they have helped us confirm that the problem is learnable with our current data and model design.

On the Raspberry Pi, we have run prototype tests to confirm that the agent can capture frames from the camera, run either a stub or simple TensorFlow Lite model, and maintain stable behaviour over time. We measure the approximate frame rate and end-to-end latency from capture to inference, and we check that the system can run for extended periods without crashing or leaking resources. We have also tested basic JSON summary generation and network communication to make sure that the Pi can reach the backend and recover from transient network failures.

6.2 Planned Testing & Evaluation Criteria

Going forward, we plan to formalise our testing around a set of concrete use cases and evaluation criteria. For functional testing, we will define user stories such as “a student signs in, creates a course and session, starts a chat, and receives AI responses,” or “a student pairs a Raspberry Pi, completes a study session, and views a focus summary,” and we will verify that each step behaves as expected, including error handling. We will treat these scenarios as regression tests that we repeat after significant changes.

For the focus model, we will evaluate it using standard classification metrics: accuracy, precision, recall, and F1-score for the “focused” and “distracted” classes. We will pay particular attention to false positives and false negatives for distraction, since misclassifying brief glances as distractions would make the system feel noisy and unhelpful. We will also examine confusion matrices to see if the model systematically struggles in certain conditions (e.g., low light or side views), and we will document these limitations.

At the system level, we will define performance targets such as acceptable AI response times for the chat (for example, first token within a few seconds) and acceptable inference speed on the Pi (sustaining around 10–15 FPS with a stable frame rate). We will also test how the system behaves under less ideal conditions: intermittent network connectivity for the Pi, temporary backend unavailability, or slow AI responses, and ensure that failures degrade gracefully (e.g., summaries are queued locally and retried).

If time permits, we would like to conduct a small, informal pilot evaluation with a few volunteers. In this pilot, we would ask participants to use the system for one or more short study sessions, then collect qualitative feedback on the usability of the interface, the clarity of the focus summaries, and whether the combination of AI tutoring and focus feedback feels useful and trustworthy. Even a small user study would give us valuable insight into how real students perceive the system and where we should focus improvements.

7.0 Risk Management

7.1 Identified Risks

We have identified several key risks that could impact the success of our project. On the technical side, there is a risk that the focus model may not achieve satisfactory performance, either because the data is too noisy or because the model struggles with real-world variability (lighting, camera angles, occlusions). There is also a risk that the Raspberry Pi may not deliver the necessary inference speed or stability if our model is too heavy or our implementation is inefficient.

From a schedule perspective, data collection and labelling for the focus model are time-consuming tasks that can easily slip, especially if we need to refine our labelling guidelines or discard low-quality data. Integration across three major subsystems (web, backend, device) is another schedule risk: interface mismatches or unexpected constraints could require redesigns late in the project if we are not careful.

Finally, there are ethical and privacy risks. Even though we designed the system to process video locally, there is still a risk of data misuse if, for example, logs or intermediate data are stored improperly on the Pi or if security on the backend endpoints is misconfigured. There is also a reputational risk if students or stakeholders misunderstand the nature of the monitoring and feel that the system is intrusive.

7.2 Mitigation Strategies

To mitigate technical risks with the model, we are starting with simple, lightweight architectures and focusing first on getting a model that is “good enough” and runs well on the Pi, rather than optimising accuracy at all costs. We also plan to iterate on the dataset incrementally: collect a modest amount of data, train and evaluate, and then refine collection and labelling based on what we learn. On the Pi performance side, we are proactively testing early with TensorFlow Lite and considering quantisation to reduce model size and inference time.

For schedule-related risks, we deliberately scheduled early integration milestones, such as a stubbed end-to-end flow from web to backend to device. This helps us surface interface issues before we have invested too heavily in any one component. We also track tasks in small units and regularly reassess priorities so that critical integration work is not deferred too long. On the data side, we are beginning collection and labelling early in the project, rather than waiting until all other components are finished.

To address ethical and privacy concerns, we have adopted a summary-only storage policy: raw video or per-frame data is processed in memory on the Pi and never transmitted to the backend. We are careful about how we store logs on the device, limiting them to aggregated summaries and necessary diagnostic information. On the backend, we enforce authentication and validation on all device endpoints, and we use HTTPS for all network communication. We also frame the system as a self-improvement tool rather than a surveillance mechanism and would, in a real deployment, accompany it with clear explanations and consent flows.

7.3 Contingency Plans

If, despite our mitigation efforts, the focus model does not reach acceptable performance, our contingency plan is to simplify the problem or the way we present metrics. For example, we could relax our expectations and focus on coarser measures (such as detecting very long distractions rather than subtle gaze shifts), or we could explicitly communicate limitations in the dashboard (e.g., confidence ranges or “low-confidence” badges). In the worst case, we could rely on a simpler heuristic-based signal (like face-presence over time) to still demonstrate the concept, even if the deep learning component is not fully mature.

If the Raspberry Pi cannot sustain the desired performance, we can adjust our design by reducing input resolution, lowering frame rates, or further simplifying the model, trading some granularity for stability. If those changes are not sufficient, we can narrow the scope of the prototype to shorter, controlled sessions instead of continuous long-running monitoring.

Should schedule risks materialise, for instance, if data collection takes longer than planned or integration proves more complex, we are prepared to de-scope non-essential features. Examples include advanced visualisations on the dashboard, secondary AI features, or more sophisticated pairing UX, in order to protect the core objective: demonstrating an integrated system that combines AI tutoring with edge-based focus summaries.

Finally, if we discover unexpected ethical or privacy concerns, we will err on the side of caution: further restrict what is logged, anonymise additional fields, or, if necessary, disable certain features (for example, any optional logging) in the final demo. In all cases, our contingency plans prioritise delivering a coherent, trustworthy prototype that aligns with our core goals, even if that means simplifying some aspects of the original vision.

8.0 Conclusion

8.1 Summary of Work to Date

So far in this capstone project, we have implemented the core pieces of the AI Study Buddy platform and validated that our overall concept is technically feasible. On the web side, we have a functional React application backed by Firebase where students can sign in, create courses, sessions, and chats, and interact with an AI Study Buddy that uses Genkit and Gemini to provide contextual responses. On the backend, we have set up Firestore as our primary data store, implemented Cloud Functions for AI chat and session management, and defined the initial endpoints needed for device pairing and focus summary ingestion. On the edge side, we have a Raspberry Pi agent that can capture camera input, run a prototype TensorFlow Lite inference loop, and log focus state over time, along with an early end-to-end path that demonstrates how a session's focus summary could be uploaded and rendered in the web app. Together, these achievements show that we are on track to meet our main objectives: providing an AI-assisted study companion and measuring focus via a privacy-preserving edge device.

8.2 Reflection on Learning and Professional Practice

Throughout the project, we have gained substantial technical experience and a deeper understanding of professional software engineering practice. Technically, we have had to design and integrate a modern web frontend, a cloud-based backend, and an embedded ML system, which forced us to think carefully about interfaces, data models, deployment constraints, and performance trade-offs. We learned how important it is to consider edge constraints (like model size and inference speed on the Raspberry Pi) early in the design, rather than treating them as an afterthought. Professionally, we have improved our ability to plan work in iterations, coordinate across sub-teams, and communicate design decisions clearly, both in code and in documentation. Working with potentially sensitive data also sharpened our awareness of ethics and privacy: we had to translate abstract principles into concrete design choices, such as keeping all video processing local and only storing aggregated focus summaries in the cloud. Overall, the project has been a practical exercise in balancing ambition with feasibility, and in building something that is technically interesting while still being responsible and user-centred.

8.3 Outlook for Final Phase

Looking ahead to the final phase of the capstone, our focus will be on strengthening the weakest links in the system, improving robustness, and refining the user experience. The main technical tasks that remain are: training and deploying a more mature TensorFlow focus model on the Raspberry Pi, completing and hardening the device pairing and summary upload flows, and finalising the focus summary dashboard so that it presents metrics in a clear and meaningful way. In parallel, we plan to invest in more systematic testing and small-scale evaluation to validate model performance, end-to-end latency, and overall usability. While there are still challenges, particularly around data quality, model tuning, and edge performance, we are confident that our current architecture and progress give us a solid foundation to deliver a coherent, working prototype that demonstrates the core idea of AI-assisted learning combined with privacy-preserving focus analytics by the end of the project.

Revised timeline

Phase	Timeline	Key Deliverables
Phase 1: Design & Setup	Oct 17 – Nov17	● Final system architecture and diagrams ● Backend and web app scaffolding created
Phase 2: Focus Monitoring MVP	Nov18 – Dec 18	● Progress Report ● Focus detection model (face/gaze)
Phase 3: AI & Document Processing	Dec 19 – Jan 19	● Document upload and embedding pipeline ● LED/sound/vibration feedback loop ● Retrieval-augmented Q&A prototype ● Citation and flashcard generation ● Live focus state visualization on web
Phase 4: Voice & Chat Integration	Jan 20 – Feb 20	● Oral Presentation ● Speech-to-text and text-to-speech working on device ● Hardware components assembled and tested ● Web chat interface connected to backend ● Real-time dual-mode interaction
Phase 5: Integration & User Testing	Feb 21 – Mar 19	● Full hardware-software integration ● Usability testing with sample users ● Focus and learning analytics dashboard
Phase 6: Finalization & Reporting	Mar 20 – April 8	● Final system report and poster ● Public demo of final prototype ● Documentation and evaluation results

Appendix

Samples

1) /device/register – Pi announces itself with claim code

Request (from Pi):

```
{
  "claimCode": "ABC123"
}
```

Response (from backend):

```
{
  "deviceId": "device_9f2a1c",
  "pairingStatus": "AWAITING_CLAIM",
  "createdAt": "2025-03-10T12:34:56.789Z"
}
```

2) /device/pairingStatus: Pi polls to see if it’s been claimed

Possible response before claim:

```
{
  "pairingStatus": "AWAITING_CLAIM"
}
```

Possible response after user claims device:

```
{
  "pairingStatus": "CLAIMED",
  "deviceId": "device_9f2a1c",
  "deviceToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

3) /device/currentSession – Pi asks if it should be recording

Response when no active session:

```
{
  "active": false
}
```

Response when there is an active session:

```
{
  "active": true,
  "sessionId": "session_47a1e2",
  "userId": "user_123",
  "courseId": "course_cs101",
  "expectedEndTs": "2025-03-10T14:00:00.000Z"
}
```

4) /device/sessionSummary – Pi uploads end-of-session summary
Request (from Pi):

```
{
  "sessionId": "session_47a1e2",
  "deviceId": "device_9f2a1c",
  "userId": "user_123",
  "startTs": "2025-03-10T13:00:00.000Z",
  "endTs": "2025-03-10T13:45:00.000Z",
  "totals": {
   "focusedMs": 2400000,
   "distractedMs": 300000
  },
  "longest": {
   "focusedMs": 900000,
   "distractedMs": 120000
  },
  "counts": {
   "distractions": 5
  },
  "avgFocusBeforeDistractMs": 480000,
  "focusPercent": 0.89
}
```

Response (from backend):

```
{
  "status": "ok",
  "summaryId": "focusSummary_8b7e3d"
}
```

References

[1]	TensorFlow, "API documentation : tensorflow V2.16.1," TensorFlow, https://www.tensorflow.org/api_docs (accessed Dec. 5, 2025).
[2]	"Firebase API reference," Google, https://firebase.google.com/docs/reference (accessed Dec. 5, 2025).
[3]	"React reference overview," React, https://react.dev/reference/react (accessed Dec. 5, 2025).
[4]	"Raspberry pi documentation," Raspberry Pi, https://www.raspberrypi.com/documentation/ (accessed Dec. 5, 2025).
[5]	T. Zhao, Y. Li, and Z. Xu, "AI-Based Focus and Attention Tracking for Online Education," *IEEE Access*, vol. 9, pp. 108245-108254, 2021

Abbreviations

API	Application Programming Interface
FPS	Frames Per Second
JSON	JavaScript Object Notation
ML	Machine Learning
ONNX	Open Neural Network Exchange
SDK	Software Development Kit
SQL	Structured Query Language
TFLite	TensorFlow Lite
UI	User Interface