GONZAGA UNIVERSITY
School of Engineering and Applied Science
Center for Engineering Design and Entrepreneurship

PROJECT PLAN

October 16, 2025

v1.0

CS26-13: Minecraft AI Companion

Prepared by:
Deepslate AI Team

Tony Nguyen

Jack Kabil

Alex Sautereau

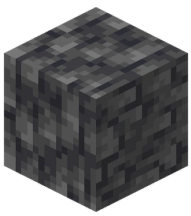
Aiden Tabarah

Reviewed by:

Peter Messina
Project Advisor

Drew Bogdan

Kevin Dang
D Squad Studios | Project Sponsor



1. Project Overview

1.1. Project Summary

Project CS26-13 aims to develop an open-source mod that provides free, integrated AI-driven capabilities for games like Minecraft. Applying AI technology to established games like Minecraft can significantly deepen existing features, enhancing players' experience. Currently, most Minecraft mods require access to a paid, external LLM provider API key, incurring costs for both multiplayer server owners and single-player users who are actively seeking free alternatives.

To achieve this, Ollama has been identified as a free and open-source alternative to proprietary AI solutions. Minecraft, a game renowned for unconstrained creativity, can greatly benefit from incorporating AI to create world-awareness capabilities in its game characters. This would make the game world feel more real and engaging while encouraging player creativity. Imagine engaging in conversations with villagers to learn about their village's history, past visitors, and personal lives, rather than merely trading random items. This level of interaction can be powered by Ollama and supported by databases that track interactions and generate dynamic text.

1.2. Project Objectives

In coordination with our sponsors, D Squad Studios, our objectives are to deliver a public, open-source Forge mod that enables world-aware villager dialogue powered by locally hosted LLMs[1] via Ollama, packaged as a signed .jar[2] with an install guide and API[3] reference.

Success criteria are:

- Working chat interaction with villagers:
  - Having an intuitive interaction with the player and establishing basic guardrails in case of failure.
  - Each conversation includes nearby context (e.g., location, structures, time of day) and can reference such facts.
- Code quality & Reusability:
  - Code to be refactorable and clean.
  - Modular packages and functions with clear interfaces conform to linter[4] and formatter[5] checks.
  - All features are supported by unit tests.
  - Having proper documentation for users and devs on how to run and maintain the project (Quickstart, Config Guide, and API Reference).
- LLM models:
  - Provide a recommendation on which LLM model works best and set it as the default.
  - Prompting and context are budgeted and capped to prevent overtokenization and ensure efficiency.
  - Set basic retrieval and chunking to ground the answer.
  - Provide adequate guidance in the system prompt to minimize contradictions.
- Persistent storage:

[1] Large Language Model: An artificial intelligence algorithm, trained on vast amounts of text data, that processes and generates human-like language.
[2] Signed .jar file: A Java Archive file that has a digital signature, allowing users to verify its authenticity and integrity, and confirming that the code has not been altered since it was signed by a trusted publisher.
[3] Application Programming Interface: A messenger that takes a request from one software application, processes it, and returns a response from another software application.
[4] Linter: A tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs.
[5] Formatter: A tool that automatically adjusts code style for consistency

- Have a simple local memory (per village, villager, or session, etc.) so that villagers can retain facts and interactions across sessions.

1.3. Project Stakeholders

Primary stakeholders are the student development team: Aiden Tabrah, Alex Sautereau, Jack Kabil, and Tony Nguyen, within Gonzaga University's School of Engineering & Applied Science. The team is responsible for design, implementation, testing, documentation, and final handoff.

The project sponsor and liaison is D Squad Studios (independent software engineering group), represented by Kevin Dang and Drew Bogdan, both GU'24. They provide product vision, technical guidance, acceptance criteria, and issue creation in GitHub.

The faculty adviser is Peter (Pete) Messina, GU'76. He is a retired IT program manager at NASA and the FAA, who ensures academic rigor, scope control, and risk management, and reviews project artifacts.

Target user communities include Minecraft players seeking immersive NPC[6] conversations without dependence on cloud infrastructure, server administrators and modpack authors who need configurable, low-latency, and privacy-preserving NPCs for public servers, and open-source developers/modders interested in extending the API, adding models, or contributing performance improvements.

Secondary stakeholders include course staff and the CEDE[7] program, who require reproducible builds, appropriate documentation, ethical use considerations, and a maintainable handoff. Communication will primarily occur through the project Discord, with separate channels for sponsors and the team. GitHub Issues/PRs will be used for decision-making and tracking, and a shared Drive will be used for design documents and meeting notes.

This project has no DAB assigned.

1.4. Project Deliverables

This project has the following deliverables:

- Villager chat generation (core mod feature): Implement generated chat when a player interacts with a villager. The chat should be able to handle at least three conversational flows[8]. This is the primary in-game behavior to be delivered as part of the mod.
- Tests: Include two key components: unit tests and functional tests. *Unit tests,* integrated with our GitHub workflow, will run on every pull request to validate key behaviors with a target of 80% code coverage. *Functional tests* will be performed after major feature releases to test functionality, performance, and security.
- Release artifact (.jar): Package the mod as a .jar and save it to GitHub so it can be downloaded and used by others. The artifact will include version notes and checksums.
- Context from nearby world structures (grounding): Enable villager chat to reference at least three types of nearby structures (e.g., other villages, etc.) to make conversations world-aware.
- (Stretch) Recent-event context: Add support for villagers to discuss at least two recent events, such as zombie attacks, weather, and players seen.
- (Stretch) Efficiency improvements: Improve efficiency by reducing at least 20% in average response time to reduce resource consumption.
- All artifacts and source code are hosted in the existing public GitHub repository: kevinthedang/minecraft-ollama.

1.5. Project Scope

[6] Non-player character: Any character in a game that is not controlled by a human player but by the game.
[7] Center for Engineering Design and Entrepreneurship: An office with in Gonzaga's School of Engineering and Applied Science that oversees the Senior Design program.
[8] Conversational flow: A distinct pattern of interaction or dialogue that AI can engage in.

For v0.1, we will deliver a locally running Minecraft mod that enables grounded, safe text chat between villagers and players using a local large language model (LLM). This build is intended for sponsor evaluation and limited user testing; it is not production-hardened.

In scope:

- Chat experience: in-game UI for Villager-to-Player text chat with at least three defined conversational flows (as listed in Section 1.4), basic safety guardrails, and graceful error handling.
- Local LLM only: integration via Ollama to run one open-weight model locally; configurable model and parameters; no cloud calls.
- World-aware prompts: prompts incorporate at least three environment signals (e.g., time of day, nearby structures, player state).
- Lightweight memory: per-villager/session memory of the last 10+ facts/interactions with simple retrieval on new chats.
- Configurability: single config file for model choice, temperature/max tokens, memory settings, and feature toggles.
- Testing and docs: unit tests covering core functional code; functional tests performed at major milestones; Quickstart, Config, and simple API docs.
- Release artifact: signed .jar published with release notes and checksums.
- Platforms/versions: Minecraft Java (Forge) on the agreed target version; desktop OS only.

Out of scope:

- Cloud capability, model fine-tuning/training, or third-party AI services.
- Autonomous agent gameplay (e.g., task-seeking player bots); focus is on dialog only, not player automation.
- Voice I/O (TTS/STT), cinematics, or new villager professions/economy systems.
- Production hardening for large multiplayer servers; v0.1 targets single-player or small private servers.
- Alternative mod loaders (Fabric/Quilt), Bedrock/console editions, or broad backward-compatibility work.
- Advanced moderation, telemetry/analytics, or localization beyond basic guardrails.
- Changes to base villager AI/pathfinding outside dialog hooks.

Interfaces and dependencies:

- Requires: user-installed Ollama runtime and chosen local model; Minecraft Java with Forge; local file storage for memory/config.
- Integrates with but does not modify: Ollama HTTP server, Forge mod loader, Minecraft player interaction APIs.

Definition of "usable product" for sponsors (acceptance criteria):

- A player can initiate chat with villagers and complete at least three conversational flows end-to-end without crashes.
- Prompts visibly adapt to at least three world/context signals.
- Villager memory recalls at least 3 recent facts/interactions within a session.
- All flows have a passing integration test; core chat/grounding code meets ≥ 80% unit test coverage; all functional tests are performed at every milestone passed.
- A signed .jar plus Quickstart and Config docs enable a fresh install to run locally without internet.

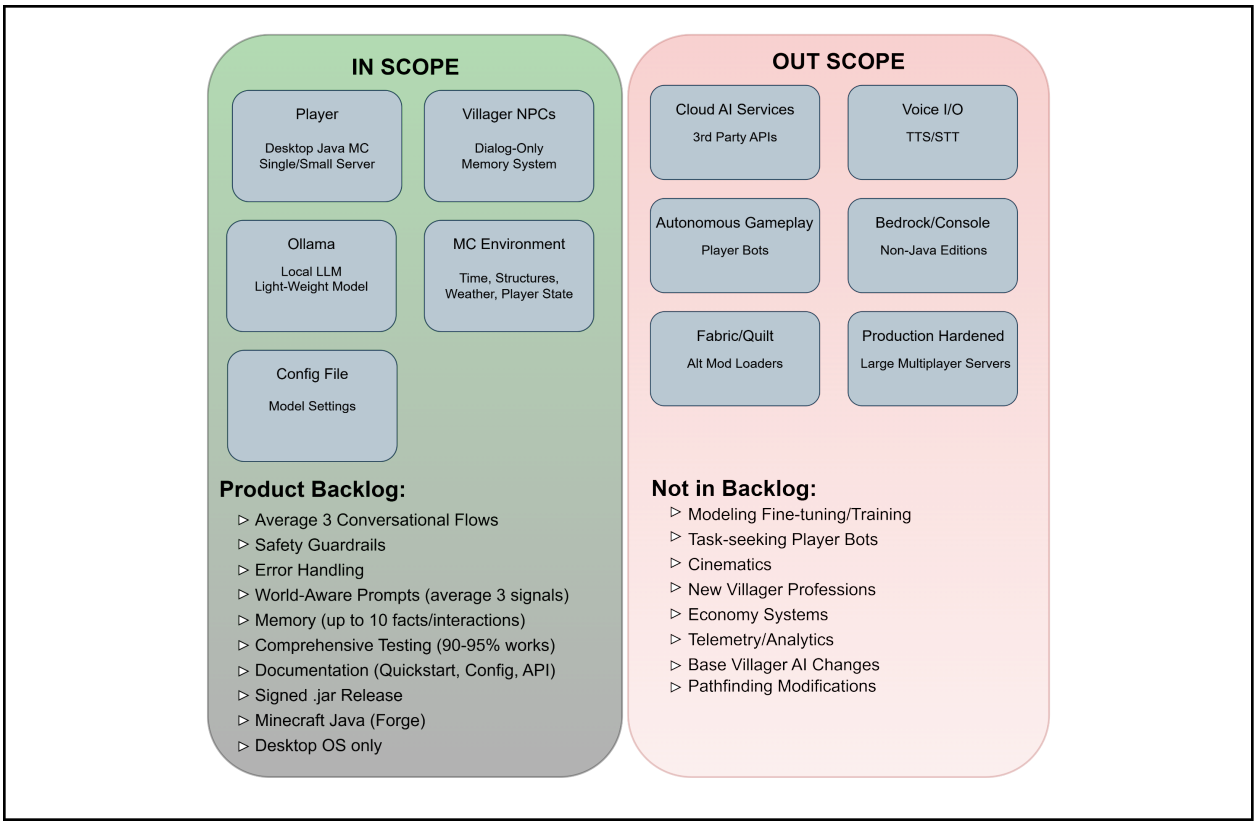


Figure 1: Context diagram detailing what components are in scope and out of scope.

1.6. Related Work

Voyager[9] is its own LLM that is a Minecraft "player" that can individually explore the world and accomplish tasks, supposedly better than existing technologies out there. This is similar to what we want to accomplish, as it's an LLM in Minecraft that can pull context from its surroundings and accomplish tasks, but it's only integrated into Minecraft players and player-related tasks. With that said, this project is a villager dialog companion: reactive and local on-screen conversations. We use Voyager only as inspiration for grounded interactions, not as a target to replicate.

The Mindcraft[10] mod builds on top of existing mods to implement LLMs into Minecraft as their own individual bots. There are a lot of things in here that we can grab directly for putting in LLMs, as Ollama works with this. This one is similar in the sense that it puts in AI for free, but doesn't put it in the villagers - they are individual players.

2. Project Requirements

2.1. Major Features

Based on discussions with D Squad Studios, as well as our target users, we identified the following major features required for a viable v0.1:

- Villager chat on interaction: This is the MVP's visible value: players and villagers get a conversational response through a chatbot-like interface.
- Local LLM integration via Ollama: Keeps inference on the player machine, avoids cloud dependence, cost issues, and aligns with sponsor goals.

[9] https://voyager.minedojo.org/
[10] https://github.com/mindcrafzt-bots/mindcraft

- Immediate and nearby world context: To be able to provide world-aware replies (e.g., village, structures, time of day) and make conversations coherent and fun.
- Base model recommendation: Recommend a default model that works best after trying several models. The model should be able to be self-hosted without any cloud dependencies.
- Testing and Code quality through unit tests + basic integration: Ensures reliability and supports public contributions. Also, ensures that the codes are correctly tested before releasing.
- Public release & documentation through signed .jar artifacts and install guides: Ensure reproducibility (for public release). Enable others to run, test, and extend the mod.

Table 1: Major Features

Feature	***Description***	***User Stories***
Villager chat on interaction	Enable conversational responses when a player interacts with a villager.	As a player, I want to talk to villagers to chat with an NPC to bring depth to my world. As a player, I want villagers to remember conversations or interactions I've had with them in the past.
Local LLM integration through Ollama	Build the mod using a locally hosted LLM (Ollama) for inference. No cloud dependency by default.	As a server hoster, I want a mod with no strings attached to add AI integration into the world.
Nearby world context	Include immediate surroundings (e.g., village/structures/time of day) in the prompts so replies reference the player's environment.	As a player, I want to ask a villager questions about the environment to find the best place to find [x] material because I don't know where to find it. As a player, I want to ask a villager questions about what different villager structures are and what they do, because I'm not sure what might be in them or how they work.
Default model recommendation	Give a recommended default model.	As a novice mod installer, I want to be provided with a recommended model that is easily deployed so that I don't have to have deep prerequisite knowledge to set up the mod. As an advanced mod installer, I want to have the ability to swap models as I wish, so that I can use a model I prefer. As a mod installer, I want to know which model is the most efficient so that I don't have many resources being taken up on my device.

		As a developer, I want to suggest the best model to the potential users of our product so that if a person installs a different model and complains about the results, it's not our fault.
Testing and Code Quality	Unit tests for core modules that run in CI[11].	As a developer, we need to know if there are any problems with the code through unit tests (low-level problems) and functional tests (high-level problems).
Public release and Docs	Package as a signed .jar artifact and publish on GitHub with an install guide, configuration instructions, and API notes.	As a player or server-host, I need an intuitive download process with a standardized and low number of steps so that set-up is easy and familiar to other mods I may have downloaded.

We also identified some "nice-to-have" features (to be our stretch goal), but decided not to include them in our Major Feature catalog. This list also consists of some GitHub Issues that have been raised by our sponsors:

- Recent-event context (zombie attacks, weather, players seen): Adds depth but is not required for MVP[12]. This feature is our stretch goal.
- Efficiency and performance: We recognize that this is a valuable and essential aspect of responsible AI usage. However, we have decided to produce a working solution first (while still keeping this point in mind), and then later improve the LLM if time permits.
- Server packaging[13]/Dev server[14] automation (GitHub issues #4, #6): This is part of the CI/CD[15]/DevOps[16] workflow rather than a core feature.
- General chat automation & server-wide LLM messaging (GitHub issues #7, #10): Not needed for villager interactions.
- Repo setup improvements & onboarding docs (GitHub issue #11): This would be an operational improvement, not needed for the core functionality to be working.
- Research documentation on villager interaction design (GitHub issue #9): Would be helpful for future user experience works, but not needed for the core functionality to work.

To determine our basic features, we synthesized our sponsor requirements (open-source, local LLMs, villager chat MVP, public release), early user expectations (quick setup, world-aware replies), and feasibility constraints (Forge API toolchain, single-player first). We then prioritized features that:

- Directly enable playable NPC conversations.
- Provide local LLM options to reduce cost constraints.
- Design the mod with the ability to extend, update, and accessibility in mind. Items that improve scale, automation, or breadth of behaviors were placed into the backlog as stretch goals.

[11] Continuous Integration: an automated process where developers merge their code changes into a central repository frequently, triggering automated builds and tests.
[12] Minimal Viable Product: The simplest, core version of a new product.
[13] Server packaging: the process of bundling the application code and its dependencies for deployment to a production server.
[14] Dev server: a local web server for developers to quickly test and run code
[15] Continuous Integration/Continuous Deployment: the automated practices and tools within DevOps that streamline the software lifecycle from code integration and testing to delivery and deployment.
[16] DevOps: a cultural and operational philosophy emphasizing collaboration and automation between dev and ops.

2.2. Initial Product Backlog

Priority scheme:

- P0 (Critical for v0.1): Minimum setup required for the mod to compile and run
- P1 (Critical for v0.1): Core interaction prototype with minimal testing. Enables the first playable demo of the villager chat UI.
- P2 (Must for v0.1): AI integration layer connecting the mod to LLM for live responses
- P3 (Nice for v0.1): Adds world context, memory, and guardrails to make dialogue coherent and immersive
- P4 (Nice for v0.1): Testing, CI/CD, performance, and stability improvements
- P5 (Later): Packaging, signing, and documentation polish for final public release

Estimate scheme (story points):

- 1=tiny, 2=small, 3=medium, 5=large, 8=epic.
- Points reflect effort/complexity, not days.

AC is the acceptance criteria.

Table 2: Initial Product Backlog

Requirement	***Description***	***Major Feature***	***Priority***	***Estimate***
R1: Forge mod scaffold & compatibility	Project builds/runs in dev; loads in a clean Forge client matching the target Minecraft version. **AC:** gradle runClient works; mod ID/metadata present; no errors in logs at startup.	Public release & Docs	P0	2
R2: Villager interaction through chat interface	Villager interaction through the chat interface. Prototype UI and input handling built in **P1**, integrated with backend in **P2**. **AC:** Player receives a streamed response; fallback message shown if backend unavailable.	Villager chat on interaction	P1-P2	3
R3: Locally-hosted LLM through Ollama	HTTP bridge from mod to a locally hosted Ollama model. **AC:** Health check runs OK; First token arrives after request, with streaming continues without gaps (returns token as LLM generates responses); end-of-stream terminator received; cancel supported; timeouts/retries handled.	Local LLM integration through Ollama	P2	5
R4: Nearby/chunk context extraction	Gather immediate world facts (e.g., biome, nearest structure, time of day, entities within N blocks). **AC:** At least three facts injected per turn; In a quick test of 10 chats, at least one answer should quote a nearby-world fact exactly as we passed it in (word-for-word).	Nearby world context	P3	3

R5: Prompting & guardrails	System prompt + lightweight instructions to keep replies grounded/concise; budget tokens. **AC:** Max context + prompt ≤ configured cap; <10% errors (contradicting the context, e.g., say day when it's night) permitted on a 50-prompt check.	Nearby world context	P3	3
R6: Model recommendation	Ship with a recommended default Ollama model. **AC:** Modular design; README lists pros/cons.	Default model recommendation	P3	2
R7: Unit & integration tests	Unit and integration tests are written alongside each new feature to ensure functionality and demo-readiness. **AC:** CI passes; core modules covered; failing bridge triggers graceful fallback.	Testing and Code Quality	P1	3
R8: Package & publish signed .jar	Build a distributable artifact and release it on GitHub. **AC:** Tagged Release with .jar, checksums, version notes; manual install verified on a fresh client.	Public release & Docs	P5	2
R9: User & developer docs	Quickstart, install, config (model switch, context), troubleshooting; API notes for bridge. **AC:** New teammate can install/run in ≤ 15 minutes following the docs.	Public release & Docs	P5	2
R10: Performance baseline	Ensure responsive UX on baseline hardware. **AC:** On a typical machine (8-core CPU, 16 GB RAM) using our recommended model, the time from clicking a villager to the reply finishing is usually 3 seconds or less, and 95% of replies finish within 5 seconds.	Testing and Code Quality	P4	3
R11: Simple local memory (opt-in)	Persist small facts per villager/village/session. **AC:** Facts survive restart; size limits; config to disable/reset.	Nearby world context	P3	3
R12: CI pipeline (build/test)	GitHub Actions builds .jar and runs tests on PRs. **AC:** Status checks required before merge; artifacts attached to releases.	Testing and Code Quality	P1	2
R13: Fresh-clone setup improvements (from #11)	Smooth onboarding on new machines. **AC:** README/CONTRIBUTING updated; clean clone builds without manual Gradle tweaks.	Public release & Docs	P5	2

R14: Server-ready artifact & smoke test (from #4)	Verify .jar loads on a minimal server. **AC:** Server boot log shows mod active; simple chat round-trip succeeds.	Public release & Docs	P5	3
R15: Error handling & offline mode	Clear messages when Ollama is not running or the model is missing. **AC:** No crashes; player sees actionable guidance.	Villager chat on interaction	P2	2

2.3. Additional Features

Priority scheme.

- Low: If a large amount of extra time is available, a feature we would like to explore.
- Medium: If extra time is available, a feature we would like to explore.
- High: A stretch goal we would love to implement and believe is attainable.

Table 3: Stretch Goal Backlog

Requirement	***Description***	***Major Feature***	***Priority***	***Estimate***
R1: Recent Event Context	Additional context support for villagers to discuss recent events that have happened to them. For example, villagers might talk about previous weather, players they have seen, or recent attacking mobs.	Nearby World Context	High	3
R2: Efficiency Improvements	Improve the efficiency of context building and chat generation to reduce resource consumption and decrease latency or server load.	Local LLM Integration Through LLama	Medium	5
R3: Villager Relationships	Additional context for villagers to have daily lives and interactions with other villagers, allowing them to have "events," knowledge, or lore about other NPCs.	Nearby World Context	Low	6

3. Design Considerations

3.1. Initial User Interface Design

The user interface will consist of two main components. The first component will build upon the existing interface used for trading with villagers (see Figure 2.1). In addition to trading, users will now have the option to click the **"CHAT"** button, which will open a new interface menu (see Figure 2.2). In this new menu, players will be greeted by the villager and presented with the option to send a message. Similar to typical interactions with AI

models, players can type a message and click Send. The message will then be processed by Ollama, which will generate and return a response to the player.

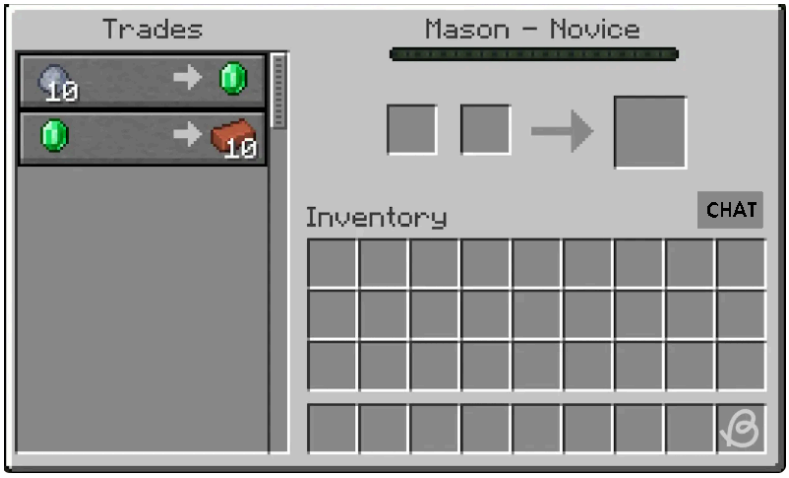


Figure 2.1: Existing villager trade menu, updated with the new "CHAT" button.

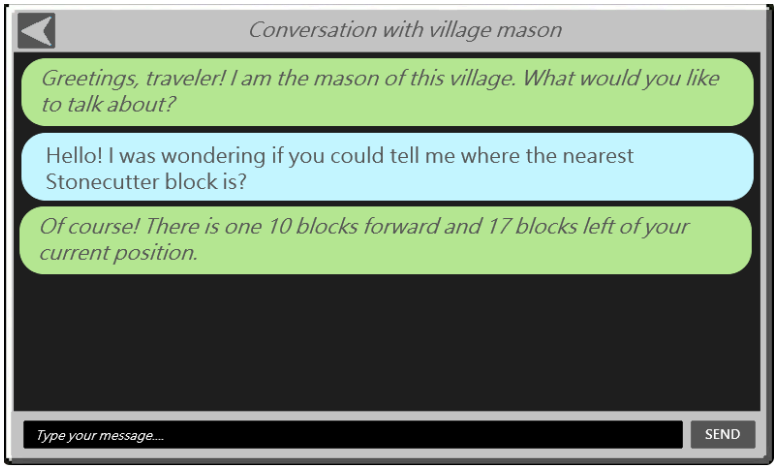


Figure 2.2: New UI menu for chatting with the chosen villager. Villager messages appear in green, and player messages appear in blue.

3.2. Initial Software Architecture

Goal: Provide world-aware villager dialogue by collecting nearby game context in a Minecraft Forge mod, submitting player prompts and context to a locally hosted LLM (Ollama) through a thin bridge, and rendering streamed responses in the in-game chat.

Major components and responsibilities:

- Minecraft mod using Forge:
    - Interaction Handler: Listens for player–villager interactions and starts the chat request lifecycle. The time from clicking a villager to the reply finishing is usually 3 seconds or less, and 95% of replies finish within 5 seconds using the recommended model, debounces repeat clicks, and exposes a cancel action.

    - Context Extractor: Gathers bounded, “chunk-level” facts (biome, nearest structure, time of day, nearby entities, etc.) within a ≤ 48-block radius, capped at 10 facts per turn with at least three facts per turn.
    - Prompt Composer: Forms the final prompt from player input + context + system prompt, enforces a configurable token budget (document default, e.g., ≤ 1024 tokens for prompt+context). Applies truncation rules and logs token counts, and rejects over-budget inputs with a concise notice to the player.
    - Client Chat Adapter: Streams model output to chat, no silent gaps more than 1.0s during streaming, provides clear fallbacks if the model is unavailable, preserves chat readability (line length limits), and supports cancel/timeout messaging.
    - Configuration & Model: Loads the recommended default model. A setup test verifies the load and return replies of both models under the specified latency targets. The README should explain trade-offs.
    - Local Memory: Maintains a bounded store per villager/village/session with at most 50 entries and at most 32 KB per villager (document defaults). Includes a toggle to disable and a command/config to reset.
    - Error Handling & Retries. Applies at most 10s request timeouts with exponential backoff (max two retries). On failure, surfaces actionable guidance (e.g., “Start Ollama” / “Select an installed model”) and never crashes the client. Logs include the error code and the elapsed time.
- LLM Bridge (Service Layer):
    - API Adapter: Minimal HTTP or WebSocket endpoint callable from the mod. Exposes a health check returning 200 within less than 200 ms locally. Supports streaming responses.
    - Ollama Client: Invokes Ollama’s generate/stream interfaces and forwards tokens as they arrive; Respects the token/length caps set by the Prompt Composer.
    - System Prompt & Policies. Applies concise system-level instructions to minimize contradictions and verbosity, documents defaults, and policy tests confirm instruction adherence on a small, fixed prompt set.
- Build/Release & CI:
    - Gradle build with unit and integration tests, GitHub Actions runs on every PR, green CI required before merge.
    - Unit tests cover the Interaction Handler, Context Extractor, Prompt Composer, and Bridge interfaces. At least one end-to-end smoke test run against a local Ollama instance or stub.
    - Publish a signed .jar with checksums and documentation.

Data and control flow:

- The player interacts with a villager. The Interaction Handler begins a chat request.
- The Context Extractor collects nearby facts under predefined bounds.
- The Prompt Composer assembles player input, context, and a system prompt within token limits.
- The mod submits the request to the LLM Bridge, which invokes Ollama and streams tokens back to the mod.
- The Client Chat Adapter renders the response in chat in real time. Local Memory may persist a small set of facts.

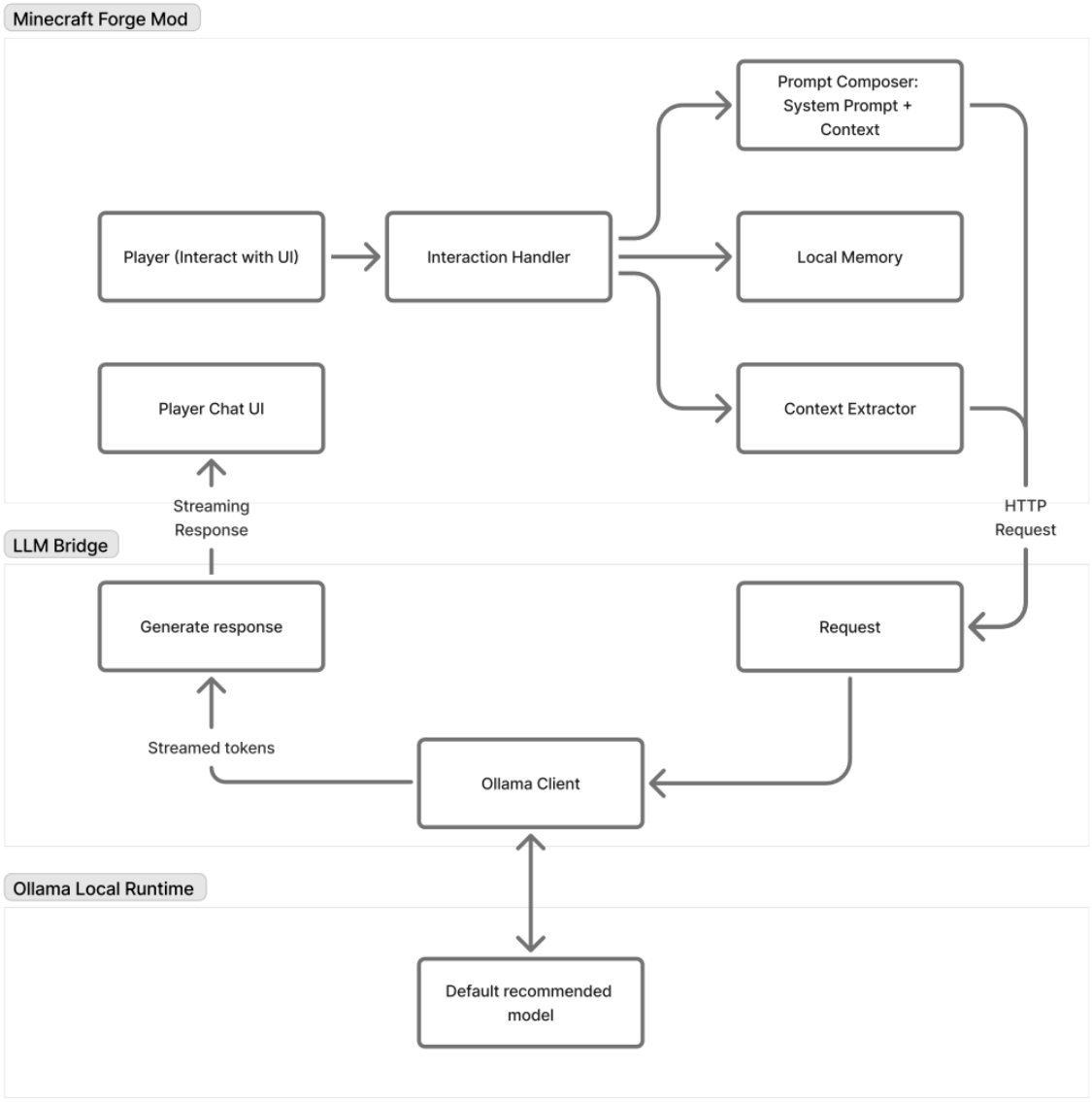


Figure 3: Architecture Design Diagram

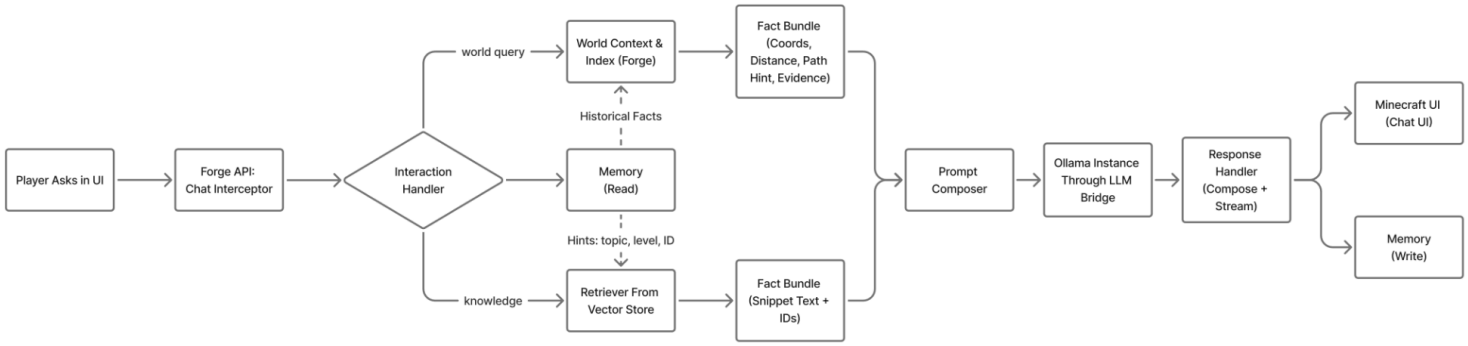


Figure 4: End-to-End Chat Flow

3.3. Development Environment, Tools, Languages, and Libraries.

Development tools & Dependencies:

- Minecraft & Forge: Minecraft (Java edition) with Forge mod loader as the integration surface for all in-game features
- Ollama: User installed, locally running LLM for open-weight model. Runs entirely on the player's machine. Ensures conversations remain private, low-latency, and free from API costs.
- Build & CI: Gradle project, GitHub actions builds and runs unit/integration tests on PR, Green CI required before merge.
- Release Packaging: signed .jar with checksums and release notes
- Configuration & Local storage: Single config file to manage parameters (token limit, model) and small local memory to recall world-context (no database or server needed)
- Repository & Docs: All source code and release artifacts will be in a public GitHub repository, as well as a Quickstart, Config guide, and API notes for future users and contributors

Languages:

- The Minecraft mod will be developed in Java using the Forge framework, which provides the necessary APIs for managing in-game interactions and handling both client and server logic.
- A lightweight LLM Bridge will be implemented as a minimal HTTP/WebSocket service callable from the mod. This bridge will expose a health-check endpoint, stream tokens in real time, and forward requests to Ollama.

IDEs, Testing harness, and CI/CD:

- The primary development environment will be IntelliJ IDEA due to its strong support for Java and Gradle projects, integrated debugging tools, automated refactoring, unit test execution, and GitHub integration.
- Unit and integration testing will be applied across core components, including the Interaction Handler, Context Extractor, Prompt Composer, and Bridge interfaces. At least one end-to-end smoke test will validate functionality against a local Ollama instance or stub. A coverage target of at least 80% is required for the core functional code that supports the major feature set. At every milestone, a functional test will be performed to ensure the product's operability.
- For CI/CD, GitHub Actions will build the project and execute tests on every pull request, with successful runs required before merge. Release artifacts will be attached to tagged versions to ensure reproducibility.
- In addition, the project will use the *Ada servers* provided by Gonzaga University's Computer Science department. These servers ensure consistent builds, reproducible test results, and adequate computing resources for integration and testing.

Expected components of Initial Design: The initial design will be structured into three major layers that together enable world-aware villager dialogue powered by a locally hosted LLM.

- Minecraft Mod Layer (Forge-based):
    - This layer manages all in-game logic, including the initiation of villager and player conversations, the collection of environmental context from the game world, prompt assembly within defined token budgets, and the rendering of live responses in the client chat interface.
    - Supporting modules handle persistent lightweight memory, error recovery, and configuration management. Each module is bounded in scope to simplify future extension and refactoring.
- LLM Bridge Layer (Service adapter):

    - Lightweight service exposing a local HTTP/WebSocket endpoint, forwards requests from the mod to Ollama, and streams model tokens back in real time.
    - Built-in health checks, policy enforcement, and system prompt controls ensure reliability and consistency in responses.
    - By separating this logic from the Minecraft codebase, the bridge maintains loose coupling between the mod and the AI runtime.
- Building, Testing, and Release Infrastructure:
    - Gradle build system and GitHub Actions pipeline form the backbone of testing and release management.
    - Automated builds validate unit and integration tests on each pull request, while end-to-end smoke tests confirm correct operation with Ollama.
    - Signed .jar artifacts with release notes and checksums are generated to guarantee reproducibility

3.4. Initial Software Test Plan

Our plan for testing will include two parts - functional testing and unit tests. We will be creating tests alongside our code. Each feature should either come with unit tests or functional tests, or building the tests is added to the backlog to be done later.

- Unit tests will be run every time a feature or branch is merged into main, guaranteeing that our code is functional.
- Functional tests will be conducted by one of the team members each time a major feature or milestone is added to ensure that all features function properly and that the deliverables made so far still work as intended.

We will have multiple types of functional tests that aim to cover both performance and security. Each one will be themed around a different feature or concern we're looking to include or address. These functional tests will also be able to address different kinds of user groups, or be aimed at testing features that only one specific user group might be affected by changes in that feature.

Unit Tests:

- The goal is to cover 80% of the code, which is ambitious, but it gives quality assurance for code quality.
- We will use GitHub workflow integrations to run unit tests for every pull request continuously.
- Example: A couple of unit tests might cover a function that finds nearby structures. These tests would validate the output for no structures, some structures, and unknown structures.

Functional Tests:

- Functional tests will primarily be functional tests where we test a specific feature from a player's perspective.
- These tests will provide a step-by-step outline of actions to take, ensuring that all edge cases are covered. Any team member can run these tests independently, and the results will be consistent with those of others.
- Functional tests will also include acceptance testing, where we might compare a server's load without the mod to its load with the mod to see how much it changes. We will set an acceptance criterion based on input from the sponsor.
- These tests will also try to hit known security vulnerabilities. For example, talking to the villager about trying to make bombs should redirect the conversation to how to make TNT, or refuse outright to give a response to that.

- Example: *Villager Chat No Structures Functional Test:*
    - Walk up to a villager in a remote location - guarantee there are no structures.
    - Click the chat button when interacting with the villager.
    - Ask “Where are we right now?”
    - Output should mention specific blocks, biomes, weather, or animals nearby.
    - Ask, “What are you doing so far from home?”
    - Output should mention being lost or some other feasible reason for not being near any structures.

4. Project Risks

Risk 1: Ollama API Changes

- *Why it is a risk:* Ollama may update its API at any time during our development cycle or after release, which could potentially break our integration. Since we do not control Ollama's development or release schedule, API changes could require significant refactoring of our code, delay our May release, or render the mod non-functional if changes occur post-release.
- *Actions to prevent / How we will monitor:* We will regularly monitor Ollama's GitHub repository and release notes to stay informed of upcoming changes.
- *Trigger events:* If Ollama releases an API update, we will assess the scope of changes. Minor changes (e.g., improved performance methods, additional optional features) will trigger an evaluation of whether upgrading is worthwhile. Significant breaking changes (e.g., fundamental restructuring of API endpoints or response formats) will trigger immediate mitigation planning.
- *Mitigation plan:* For minor updates, we will evaluate the benefits and decide whether to upgrade based on performance improvements or new features. For major breaking changes, Plan B is to continue using the older API version and clearly document this requirement for users. If the benefits of the new API are significant, Plan C is to allocate development time to refactor our integration layer to support the new API version.

Risk 2: Unfiltered/Irrelevant AI Responses

- *Why it is a risk*: The AI model may generate inappropriate or irrelevant content during gameplay, which could negatively impact user experience or introduce offensive material into the game environment. This could damage the project's reputation and make it unsuitable for specific audiences.
- *Actions to prevent*: We will implement guardrails, including prompt engineering techniques to guide the AI toward appropriate and relevant responses, and content filtering mechanisms to catch problematic outputs before they appear in-game.
- *How we will monitor*: The development team will conduct functional in-game testing throughout development, deliberately attempting to trigger inappropriate or irrelevant responses to assess the effectiveness of the guardrails. We will also test various edge cases and unusual player inputs.
- *Trigger events*: If testing reveals that inappropriate or irrelevant content is getting through our guardrails with any regularity, this will trigger immediate mitigation efforts.
- *Mitigation plan*: Plan B is to fine-tune our guardrails by adjusting prompt engineering techniques, implementing stricter content filters, or adding additional validation layers. Plan C is to implement user-facing warning messages that inform players about inappropriate messages.

Risk 3: Third-Party Model Provenance & Data Handling

- *Why it is a risk*: Even though our design is “local LLM only” (no cloud calls by default), users may load third-party model files. Some models have unclear training-data provenance, evolving

licenses, or unexpected behavior. We recognize there can be concerns with certain models (e.g., DeepSeek as an example of a third-party model).

- *Actions to prevent*: We will run in "local-only" mode (disable remote pulls/telemetry, block outbound traffic), and sandbox the runtime with least privilege.
- *How we will monitor*: We will run a startup check, watch for any outbound connections during play, and regularly probe with edge-case prompts to see if the guardrails hold, while tracking upstream version changes.
- *Trigger events*: If a model fails any checks, shows any unexpected outbound connection, changes license or terms, or our guardrail failure rate crosses the threshold, we will act according to the Mitigation Plan.
- *Mitigation plan*: Plan B: revert back to the last known-good model, tighten filters and sandbox, then retest. Plan C: remove the model from our testing list and publish a short migration note.

Risk 4: Key Member Missing from a Scheduled Meeting

- *Why it is a risk*: A member missing from a key meeting hinders the team's ability to communicate with each other, communicate with stakeholders, and communicate with our advisor. Unplanned activities might cause a team member, an advisor, or a sponsor to not show up to a meeting. We should be prepared for this risk.
- *Actions to prevent*: Every member should have knowledge about what and why others are working on what they are working on.
- *How we will monitor*: Pull request reviews ensure there are two people looking over every feature. If there are frequent questions being asked about how something works after it's been merged, there might be a problem.
- *Trigger events*: Communication via Discord about not being able to make a meeting, or 10 minutes into a scheduled meeting, the key member doesn't show up without any communication.
- *Mitigation plan*: If the meeting would not be productive without the member, the meeting can be revamped into a work meeting, or be reconfigured to accomplish a different goal that is doable with the people involved. Otherwise, the meeting could be rescheduled for later in the week or skipped until the next weekly meeting.

Risk 5: Member Conflict 50/50

- *Why it is a risk*: The composition of the team is 4 people - that means any agreement / disagreement split 50/50 cannot be solved democratically. If enough discussion goes by and nobody seems close to changing their mind, there must be a protocol to solve the problem.
- *Actions to prevent*: Facilitating conversation and discussion, prioritizing leading arguments with data/experience, and not with emotion. Get second opinions from other computer science groups or our advisor/sponsors.
- *How we will monitor*: If a discussion feels like it's going nowhere in coming to a decision, someone is able to say "Can we call a vote?" and will start a formal process.
- *Trigger events*: If a vote is called and is split 50/50 down the line, further steps must be taken.
- *Mitigation plan*: Postpone the decision, and ask the advisor for input into the discussion, and eventually a vote as well.

5. Initial Product Release Plan

5.1. Major Milestones

Table 3: Major Milestones

Milestone	*Description*	*Target Completion Date*
M0: Repo & CI Online	Coding environment setup. Forge scaffold builds and runs successfully; Gradle and GitHub Actions pass (R1, R12).	Week of Nov 2, 2025
M1: Chat UI Prototype	Villager chat UI & interaction handler; placeholder responses with graceful fallbacks (R2, R7 partial).	Week of Nov 17, 2025
M2: LLM Bridge and Streaming Ability	Local Ollama bridge with health check, streaming tokens, cancel/timeout handling (R3, R15).	Week of Nov 30, 2025
M3: World Context and Guardrails	Extract 3+ nearby facts/turn; prompt composer with token caps; basic contradiction checks (R4, R5).	Week of Jan 26, 2025
M4: Model Rec + Light Memory	Investigate and recommend a default model with trade-offs; create a per-session memory (R6, R11).	Week of Feb 9, 2026
M5: Performance Baseline	Meet UX target on baseline hardware; functional performance test (R10).	Week of Feb 23, 2026
M6: Release	Signed .jar + checksums; Quickstart,Config,API notes; minimal server test (R8, R9, R14, R13).	Week of Mar 23, 2026
M7: v0.1 Public Release & Handoff	Final polish, docs, and sponsor sign-off; archive artifacts; maintenance notes delivered.	Week of Apr 20, 2026

5.2. Initial Sprint Releases

Table 4: Sprint Release Plan

Sprint Date	*Spring Goal*	*Backlog*	*What we will demo*
Oct 13 – Nov 2, 2025	Repo & CI online; mod boots in clean Forge client	R1 Forge scaffold; R12 CI pipeline	Build runs locally and in GitHub Actions; green unit-test harness; startup logs are clean, and the uploaded build artifact is present.
Nov 3 – Nov 16, 2025	Chat UI prototype and basic tests	R2 (P1) Villager interaction; R7 tests	Click-to-chat with a placeholder response and graceful fallback if the backend is down; PR checks are passing.
Nov 17 – Nov 30, 2025	LLM bridge with streaming. Cancel and timeout handled	R3 Ollama bridge	Live-streamed tokens, health check, cancel/timeout behavior on demo prompts.
Dec 1 – Dec 12, 2025	Buffer for final exams and end-of-semester reports	N/a	Presentation 2 (Dec 3) Fall status report (Dec 4)
Dec 13, 2025 – Jan 12, 2026	Winter break	N/a	N/a
Jan 13 – Jan 26, 2026	Develop world context capability and prompt guardrails	R4 nearby/chunk context; R5 prompting & caps	Replies that quote at least one injected world fact; contradiction error rate spot-check; first functional test pass.
Jan 27 – Feb 9, 2026	Model recommendation and implement a simple memory	R6 recommended model; R11 simple local memory	Default model recommendation with pros and cons;

			per-villager/session fact recall; functional test round 2.
Feb 10 – Feb 23, 2026	Performance baseline	R10 performance baseline	Measured latency on baseline hardware; performance notes in README.
Feb 24 – Mar 9, 2026	Generate documentations	R9 user/dev docs; R13 fresh-clone setup	A new teammate installs and runs with ease; documentation is complete.
Mar 10 – Mar 23, 2026	Server artifact and smoke test	R14 server-ready artifact	Minimal server boot with mod active; simple villager chat round-trip; deployment dry-run.
Mar 24 – Apr 6, 2026	Error handling & offline mode; polish	R15 error handling/offline; bugfixes	Demo degraded/offline behaviors and recovery paths; stability improvements; functional test round 3.
Apr 7 – Apr 20, 2026	Public release and handoff	R8 signed .jar; finalize R9/R14	Public GitHub release with signed .jar, checksums, Quickstart, and Config documentation ready; deployment guide finalized; handoff checklist.

6. Maintenance Considerations

Our project may require periodic maintenance to address updates to the Ollama API, Forge framework, or Minecraft versions that change Java sources (currently built for Java 21). During development and immediately post-release, our development team will handle maintenance, while future maintenance will transition to employees at D Squad Studios. Both parties will require a medium to advanced understanding of Java, Forge modding, and API calling/AI model interaction. The most critical maintenance area will be the bridge between Minecraft and the AI model, as AI models evolve and their APIs change over time. To support future maintenance, we are implementing unit tests and functional tests documentation that explains how the system works, enabling future maintainers to understand and modify the codebase as needed. Updates and patches will be released through our GitHub repository with detailed changelogs documenting all changes.

7. Project Management Considerations

Team cadences, check-ins with the sponsor, roles, and tools are listed below:

Cadence & Meetings

- Weekly hurdle & Advisor updates: Tuesday 5:30 - 6:30 PM (Zoom): Quick team update and sync up with Advisor.
- Work session: TBD based on Sprint needs.
- Bi-weekly Sponsor sync: Tuesday 5:30 - 6:30 PM (Zoom): Feature basic demo, feedback, and confirm next steps.
- Additional Project Team call on Zoom/Discord over the weekends if needed.

Stakeholder Updates & Visibility

- Monthly demo package: short demo during meeting with Sponsors; kept release notes current.
- Tracking: GitHub Projects board maps to Section 5 sprints; milestones kept current.
- Each feature should be worked on in a separate branch and tied to a GitHub Project issue. Thorough testing needs to be done before merging to main.
- Decision log & minutes: Meeting notes in /docs/meetings/. Meeting recordings are available on Zoom.

Roles & Ownership

- Aiden Tabrah: World Context & Memory
    - Per-villager/session memory; context extractor (biome/structures/time/entities).
- Alex Sautereau: Guardrails & Performance
    - Token budgets & system prompt; performance baselining & profiling.
- Jack Kabil: Client UX & Interaction
    - Villager chat UI/flows; streaming/cancel UX.
- Tony Nguyen: LLM Setup & DevOps
    - Ollama bridge (health/streaming/timeouts); CI/CD, releases, docs.
- Each feature ships with tests by the author, and PR is reviewed by someone outside that area.

Collaboration, QA, and Release

- Source & CI: GitHub (Issues/PRs/Projects); GitHub Actions (build, unit, integration).
- Communication: Discord (daily), Zoom (sponsor/advisor), Email (major status updates; communicate with Advisors/Sponsors).
- Docs: Markdown in /docs (Quickstart, Config, API kept release-ready).
- Testing: JUnit + fixtures; functional test passes at major milestones.
- Releases: Signed .jar and checksums on GitHub Releases; CHANGELOG for user-visible changes.

Change Control & Escalation

- Change requests: file as GitHub Issue labeled proposal-change; discuss in sponsor sync; accepted changes reflected in backlog and next sprint goal.
- Risk/blocker escalation: if a blocker lasts more than 48 hours, escalate at the next daily window; if scope/schedule/quality is at risk, schedule a quick “emergency meeting” to review and adjust the plan.

Definition of Done (per feature)

- Code merged with passing CI; tests (unit/integration) included.

- Docs updated.
- Demo scenario recorded or reproducible steps provided.
- Performance/guardrail checks passed when relevant.

Team Member Bio

Aiden Tabrah

Majors: Computer Science (B.S.), Concentration in Data Science

Education:

- *Mountainside High School* | 2018 - 2022 | Beaverton, OR | **4.1 GPA**
- *Gonzaga University* | 2022 - 2026 | Spokane, WA | **3.85 GPA**
    - President's List 2022 - 2026.
    - Most relevant upper-division courses: Data Structures, Software Development, Database Management, Intro to Data Science, Computer Organization, UI/UX, Ordinary Differential Equations, and Assembly.

Related Work Experience & Software Projects:

- ***Skillsforge AI* | *Lead Founding Developer* | Beaverton, OR | 2025 - Present | JavaScript:**
    - Created a codebase from scratch, utilizing OpenAI and Gemini APIs to create an AI educational website.
    - Managed and designed a PostgreSQL database, dev workflows, and worked effectively and efficiently with AI to review and develop code with human oversight.
- ***WiSA* |*Full-Stack Software Intern* | Beaverton, OR | 2023 - 2025 | Python, Kotlin:**
    - Independently designed and developed a Wi-Fi signal analyzer app, integrated into a larger diagnostic Android app, enabling customers to assess product performance.
    - Utilized REST APIs, SQL, databases, OOP principles, Git, and Agile methodologies to deliver robust software solutions.
- ***Independent Video Game Developer* | *Roblox Games* | 2020 - Present | Lua, 3D modeling:**
    - Designed and developed four games on Roblox Studio, managing full project life cycles and leading a team of peers to deliver engaging gaming experiences.
- ***Project Developer / Project Lead* | *Gonzaga Hackathon* | Spokane, WA | 2022 - 2023 | Python:**
    - Directed a 4-member team to create an AI simulation where agents interacted dynamically based on environmental factors.
    - Secured 2nd place consecutively, recognized for innovative and functional project design.

Skills:

- My skills are mentioned above, but I'll provide a summarized list here:
    - Git, SQL, REST APIs, databases, OOP, Agile, Lua, C++, Python, JavaScript, Typescript, 3D Modelling, AI workflows, OpenAI/Gemini APIs, Vercel, supabase, pinecone, RAG-assisted AI, data structures.

Other interests / Personal information:

I've always been in love with the musical arts – I play piano, tuba, and trombone! I play in the Gonzaga Wind Ensemble, jazz bands, and jazz combos. I enjoy reading sci-fi/fantasy books in my free time, watching TV shows or movies with friends, or playing video games. I also try to stay active, so I go to the gym regularly, and I've started to go to rock climbing gyms with friends!

Jack Kabil

Majors: Computer Science (B.S.)
Minors: Applied Mathematics

Education:

- *Arapahoe High School* | 2018 - 2021 | Centennial, CO | **4.2 GPA**
    - Graduated high school a semester early.
- *Gonzaga University* | 2022 - 2026 | Spokane, WA | **3.91 GPA**
    - President's List 2022 - 2026.
    - Most relevant upper-division courses: Algorithms & Abstract Data Structures, Software Development, Operating Systems, Database Management, Data Science Algorithms, Organization of Programming Languages, and Applied Linear Algebra.

Related Work Experience & Software Projects:

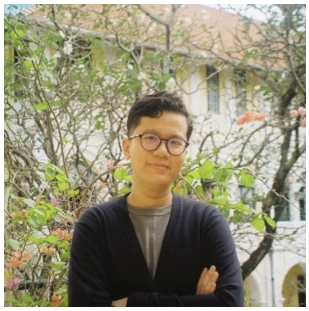
- ***American Cementing* | *Full-Stack Intern* | Centennial, CO | 2025 - Present | C#:**
    - Used Object Relational Mapping tools to process, manipulate, collect, and present data in multiple formats (tables, graphs, exportable reports).
    - Designed new features for the system, including Bill of Lading Generator for shipping, a new Supplier setup system, and an R&M Parts system rework for better functionality.
- ***Gonzaga University* | *Teaching Assistant* | Spokane, WA | 2024 - 2024 | Python:**
    - For ENSC 301: Programming for Engineers, acted as a grader/code review for engineering students learning how to code in Python.

Skills:

- A summarized list of my skills includes:
    - **Coding Skills:** Git, SQL, ORM, OOP, ASP.NET Web Forms, C#, Java, C++, Python, LaTex, Jupyter Notebooks, Excel, and applied mathematics.
    - **Personal Skills:** Time management, teamwork, communication, and being personable.

Other interests / Personal information:

I am easily hooked on anything sci-fi, fantasy, or just generally "nerdy." I play Magic: The Gathering, mostly in the Commander format, which I got into after a professor explained it was Turing complete—I love watching the games play out like giant simulations. Growing up in Colorado, skiing has been a part of my life since I was four, and it is still one of my favorite ways to spend time outdoors. I also enjoy trying new foods and, when time allows, playing video games with friends.



Tony Nguyen

Majors: Computer Science (B.A.), *Data Science*
Business Administration, *Finance*

Education:

- *Gonzaga University* | 2021 - 2026 | President's Honor List | Spokane, WA.

Related Work Experience & Software Projects:

- ***Institute for Informatics & Applied Technology* | *AI Research Assistant* | Spokane, WA | 2025 - Present | Tech: Python, Azure AI Foundry, LangChain, RAG.**
    - Benchmarked 3 LLMs on CVE reports and built a three-stage RAG pipeline using AI agents to improve insightfulness and verifiability.
    - Implemented automated evaluation and observability using Azure AI Foundry. Researched semantic embeddings to address conflicting sources.
- ***KPIM Vietnam* | *Data Engineer Intern* | Hanoi, Vietnam | Jun 2025 - Sep 2025 | Tech: Python, SQL, Microsoft Fabric, Power BI, Git.**
    - Designed and maintained 10+ pipelines across two big-data projects.
    - Built reusable tooling and documentation to enable self-serve financial and operational data for six hotel properties.
- ***KONE Elevators & Escalators* | *Data Engineer Intern* | Singapore | Jun 2024 - Aug 2024 | Tech: AWS S3, Tableau, SAP, Salesforce, Excel.**
    - Integrated 20 years of data from five countries into a centralized model.
    - Built Tableau dashboards to monitor customer behavior, service KPIs, and revenue in real time. Built ETL pipelines using AWS S3.
- ***TecAlliance Vietnam* | *Data Science & ML Intern* | Ho Chi Minh City, Vietnam | Jun 2023 - Aug 2023 | Tech: Python, Neo4j, YOLO, SVG Parsing, Git.**
    - Engineered parsers for wiring-diagram SVGs to create structured datasets for computer vision and ML pipelines. Deployed Neo4j to model complex relationships, improving query speed by 80%.
- ***Gonzaga University* | *Teaching Assistant* | Spokane, WA | Aug 2024 - Present.**
    - Tutored and graded assignments for three CS courses (~150 students) at GU.
- ***Mortality Risk Estimator* | *Project Lead* | Spokane, WA | Spring 2025 | Tech: Python, scikit-learn, PyTorch, pandas.**
    - Built a neural network to predict individual mortality risk from health attributes to inform insurance underwriting.

Skills:

- **Coding:** Python (scikit-learn, PyTorch, Flask, Streamlit), SQL, C++, Java, Git, Linux.
- **Data Engineer Tools**: Tableau, Power BI, Microsoft Fabric, Salesforce CRM, Docker, Google Cloud Suite, AWS Suite, ETL.
- **AI / LLM:** RAG pipelines, LangChain, CrewAI, Semantic Kernel, Hugging Face Transformers, MCP, AI Agents.

Other interests / Personal information:

I enjoy running and exploring Spokane outdoors. I am passionate about travel, cooking, and music discovery, with a special interest in Taylor Swift, Gracie Abrams, and Sabrina Carpenter's work. I'm also active in peer mentorship and involved in the GU Vietnamese Club VSA.

Alex Sautereau

Major: Computer Science (B.S.), *Concentration in Software Development*

Education:

- *Gonzaga University* | 2022 - 2026 | Spokane, WA.
    - Most relevant upper division classes: Algorithms & Abstract Data Structures, Software Development, Operating Systems, 3D Graphics, Web Development, Database Management, Data Science Algorithms.

Related Work Experience & Software Projects:

- ***FASPS* | *Fullstack Intern* | Seattle, WA | 2025 - Present | Tech: Python, SQL, JavaScript, HTML/CSS.**
    - Maintained and optimized servers and network infrastructure (Windows servers), using Python for automated network diagnostics.
    - Improved SQL database maintenance for better data backup/recovery.
    - Designed and implemented subpages for the school's official website using UI/UX principles in HTML/CSS and JavaScript.
- ***ArenaNet* | *Game developer partner* | Seattle, WA | May 2023 - 2024 | Tech: Java, C#, API integration, Git, Unity, Blender.**
    - Collaborated with a game development studio to deliver continuous updates and patches, practicing CI/CD pipelines and release management.
    - Balanced client requests, deadlines, and performance constraints, adapting features to meet both technical limits and design goals.
    - Designed 3d models and 3d scenes in Blender/Unity.
    - Created Unit, Performance, and Security tests to ensure projects were efficient and reliable.
- ***Independent Game developer* | *Godot/Unity* | May 2022 - Present | Tech: 3D modeling, Godot, Unity, Three.js, Node.js, Game design**
    - Managed and designed various smaller and larger-scale video games across different platforms and languages (Java, JavaScript, C#), ranging from 3D galactic space shooters, educational history facts guessing games, FPS, and horror puzzle solving.
    - Focuses on user experience and performance optimization, held closed betas for beta testing.

Skills:

- Most of my skills have been mentioned, but here is a summary:
    - Java, JavaScript, C#, SQL, MongoDB, Bash, HTML/CSS, UI/UX design, Unity, Godot, Blender, Game design, Performance testing and optimization, CI/CD pipelines, 3d modeling.

Other interests / Personal information:

I love traveling and outdoor activities, such as hiking, swimming, camping, and kayaking. I try to spend most of my time outside when I can. I also really enjoy different kinds of arts, particularly music (I play guitar and piano), as well as cinematography. I'm passionate about making creative games that impact others, often trying to combine my other interests, such as cooking or animals, into my passion projects. I was also born and raised in Paris, France.