How AI Built This System

This project was developed using an AI-first workflow in which generative AI systems acted as the primary development agents, while I focused on system design, evaluation, supervision, choice making, and verification. Rather than manually writing code in a traditional software engineering process, AI tools were used to generate backend and frontend code, propose system architectures, design prompts, debug errors, and draft documentation. My contribution consisted of defining the system's goals, constraining AI behavior, testing, debugging errors, and evaluating whether the resulting system met the project's epistemic and functional requirements.

Two primary AI tools were used throughout development: ChatGPT and Google Gemini. ChatGPT was used extensively during the design phase to reason about the overall architecture of the system, explore different approaches to AI-assisted research workflows, design the multi-stage epistemic pipeline, and critique both system behavior and AI-generated code. Google Gemini was used in two different ways. One was as the main AI coder (as ChatGPT was found to be more outdated in what tools it referenced). The second way was as the backend inference model (in other words the API) and was responsible for generating the actual research outputs produced by the final product. Gemini was selected because it supports flexible API-based access, structured JSON output, and was suitable for repeated multi-stage calls within a single research run.

AI was used to generate code and also to generate and refine the prompts that define each stage of the REFLEX pipeline. Instead of a single prompt, the system uses multiple role-specific prompts that constrain the model's behavior at each stage. For example, the intent-parsing stage uses a prompt that explicitly instructs the model not to answer the research question, but instead to extract assumptions, define terms, resolve ambiguities, and establish scope. Later stages deliberately introduce adversarial behavior, such as a critique stage that is instructed to assume claims are incorrect unless justified. These prompts were iteratively refined through experimentation, with AI suggesting revisions and the human evaluating whether those revisions improved clarity, rigor, or error detection.

Although AI generated the majority of the code, meaningful technical knowledge was still required to successfully build and run the system. Understanding APIs and HTTP request lifecycles was necessary to connect the frontend and backend and to diagnose issues such as failed requests and CORS errors. Familiarity with JSON was required to design structured schemas for model output and to ensure that AI-generated responses could be safely parsed and rendered by the frontend. Basic deployment concepts, such as Python virtual environments, dependency management, environment variables, and running local servers, were also required in order to execute the system reliably.

At the same time, AI significantly reduced the need for traditional low-level programming skills. The human developer did not need to manually write boilerplate backend code, remember framework-specific syntax, or design frontend layouts from scratch. Instead, AI handled these implementation details, allowing the human role to focus on higher-level concerns such as system correctness, scope control, and epistemic reliability. This shift illustrates how AI changes the nature of technical work from direct implementation toward supervision and evaluation.

AI failures occurred at multiple points during development. The model occasionally hallucinated technical details, such as suggesting deprecated libraries or unsupported API parameters. In testing research outputs, the model frequently produced claims that were confident but insufficiently supported, particularly in open-ended or contentious domains. Additionally, broad research questions sometimes caused the system to generate excessively large outputs, leading to rate-limit errors or resource exhaustion. These failures highlighted the necessity of human oversight and iterative refinement.

Correctness was evaluated through several mechanisms. The system itself enforces internal checks by requiring claims to pass through an adversarial critique and revision process. Outputs were also compared against a standard single-prompt “consumer” chatbot response to assess whether REFLEX **meaningfully** improved the following metrics of clarity, caution, and transparency. Finally, manual human inspection was used to verify that assumptions were explicit, confidence was appropriately calibrated, and limitations were acknowledged.

Overall though, I found working with AI to be quite pleasant. What was new this time was working with both ChatGPT and Gemini simultaneously and passing them messages. It was very interesting to see how effectively they communicated compared to how I usually communicate with chat bots.

Using AI blindly without supervision would pose significant risks. Fluent language can obscure uncertainty, hallucinations may appear authoritative, and subtle technical errors can silently undermine system behavior. This project demonstrates that while AI can act as a powerful development agent, it requires careful constraint, monitoring, and evaluation—particularly in research-oriented applications where correctness and transparency are critical.