Sudoku Sage: An AI-Powered Interactive Sudoku Assistant

Aidan Gillespie
Vladimir Serov
Drew Phelps

November 25, 2025

Abstract

This project proposes an interactive Sudoku platform enhanced with AI-driven assistance. The system allows users to play Sudoku in a clean web-based interface while requesting hints and explanations powered by ChatGPT. The goal is to combine traditional puzzle-solving with personalized real-time guidance that helps players understand correct moves rather than simply revealing answers. Our implementation integrates a FastAPI backend, a custom front-end board, and an AI reasoning component that interprets the current state of the board to generate helpful suggestions. This approach not only improves the learning experience for new players, but also creates a scalable foundation for premium features such as advanced tutoring, difficulty analysis, and subscription based access to more powerful AI models. The proposed platform demonstrates the business potential of combining casual gaming with intelligent assistance to create a more fun, engaging experience.

1 Solution and Methodology

Our solution is a web-based Sudoku system built with a custom frontend using vanilla HTML, CSS, and JavaScript. This is paired with a FastAPI backend that handles generating puzzles and AI-generated hints. The frontend renders the interactive 9×9 Sudoku grid, tracks user inputs, and sends the current board state and user questions to the backend through a simple REST interface.

The backend receives these requests, validates the board string, and constructs a structured prompt that includes the current and solved puzzle state. This prompt is then processed through LangChain, which we use as the orchestration layer to manage prompt formatting, insert board context, and maintain a lightweight conversation history. LangChain routes the request to the GPT-5-nano model, chosen for its fast reasoning performance on constrained tasks such as producing Sudoku hints.

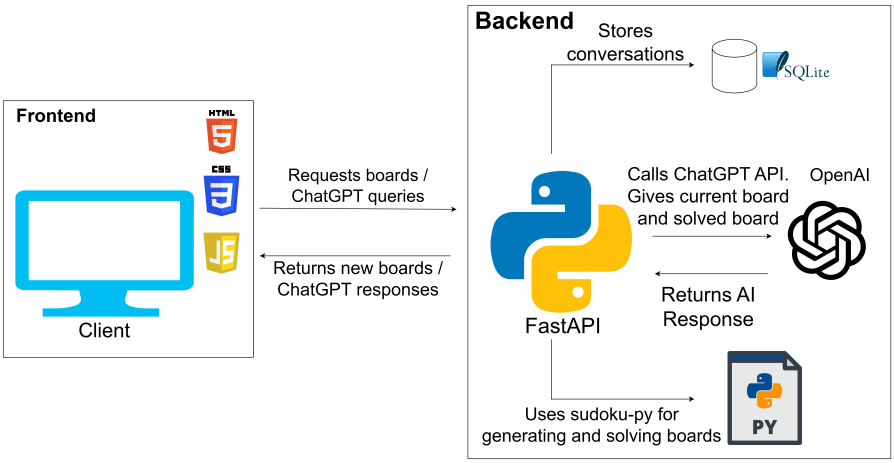

Figure 1: System Architecture

2 Implementation Details

The system is implemented using a lightweight client–server architecture. The Sudoku board displayed to the user is not generated in the browser; instead, the frontend requests a freshly generated puzzle from the backend, which uses the `py-sudoku` library to create a new valid 9×9 board. The backend returns the puzzle as an 81-character string, and the frontend stores this string as the canonical board state. The backend also stores each suggested move along with the hint and the specific reasoning behind it, allowing us to analyze model performance over time and evaluate how results might change under updated system prompts or future model upgrades.

User interaction with the grid is implemented through simple click-and-type input. When a cell is selected and the player types a number, the corresponding character in the board string is updated. Basic validation is supported, though the system currently assumes that the most recent change is responsible for any conflict, and no visual conflict markings are displayed on the board.

All communication between the frontend and backend is done through two POST endpoints. The `/board/new/` endpoint creates and returns a new Sudoku puzzle using `py-sudoku`. The `/ai/query` endpoint is responsible for generating AI hints. The frontend sends the current board string along with the accumulated chat messages, and the backend forwards these to a LangChain `ChatOpenAI` chain using the GPT-5-nano model. The backend then appends the model's response to the message list and returns the updated conversation to the frontend for display.

By keeping the frontend minimal and offloading puzzle generation, validation, and AI reasoning to the backend, the system maintains a clean separation of duties. This design makes it easy to adjust or extend the AI behavior with-

out modifying the user interface, and keeps the browser-side code simple and efficient.

3 Experiments, Tried Approaches, and Abandoned Ideas

3.1 Model Experiments

We evaluated several modeling approaches to determine the best balance between accuracy, interpretability, and runtime cost. Our experiments progressed through three broad stages:

1. **Small-to-Medium LLMs (Proof-of-Concept).** Early prototyping relied on smaller instruction-tuned models to test prompt formats. These models offered low latency and minimal cost, but frequently produced suggestions that violated Sudoku constraints (for example, proposing duplicate digits within a row).

2. **Large LLMs with Structured Input.** To improve consistency, we evaluated larger models (GPT-4 and GPT-5) and changed the prompt to cover more edge cases and input formats, listing each allowed logical technique. Larger models produced more useful, human-readable explanations; however, the large prompt size (4-5 pages) made the model take significantly more time to respond, and increased costs.

3. **Hybrid: Py-Sudoku + GPT-5.** The production approach adopted a hybrid design: the python library py-sudoku that generates the boards also gives the correct answer as a string for verification purposes. We provide this as input to the LLM, which is tasked with explaining the techniques used to solve each step, teaching the user. This separation of roles substantially reduced invalid hints and allowed the LLM to focus on explanation quality rather than search correctness.

Across these stages, we measured latency, the frequency of illegal suggestions, and the clarity of qualitative explanations. The hybrid architecture, when using GPT-5, produced the most consistent results in our prototype, yielding a large reduction in illegal suggestions while preserving the LLMs explanatory strengths.

3.2 Prompting Experiments

We systematically explored prompt-engineering strategies to elicit concise, educational hints rather than full solutions. The dimensions we varied were: (a) input representation (plain text grid vs. 81 character string), (b) level-of-detail constraints (one-step hint vs. multi-step plan), and (c) prompt token amount.

Key observations:

- **Structured input helps;** Prompts that supplied an explicit list of candidate cells, possible candidates per cell, or a small JSON board consistently reduced hallucinations and improved positional accuracy.
- **Few-shot exemplars.** Including 1–3 short exemplars showing the desired hint format and reasoning style reduced variance in responses and made the outputs easier to render in the UI.
- **Token Usage.** Our original prompt was around 4 pages long and 1,232 tokens and included some unnecessary information, which was removed in the new version. The new version shortened the prompt to 2 pages and 625 tokens and significantly reduced thinking time. This led to less cost per prompt and faster results.

Taking these findings together, our production prompt uses a structured board representation, a single example of the desired hint style, and a few output guidelines.

3.3 Hint Style Experiments

We experimented with three hint styles to assess their intellectual value and user preference.

1. **Action Hints.** Short, actionable hints that identify a single cell and the digit to place (e.g., "Place 7 in row 4, column 4 because it is the only candidate in that block"). These are efficient but can encourage dependence.
2. **Guiding Hints.** Hints framed as questions or mini-tasks (e.g., "Which cell in block 4 now has only one possible candidate?"). These support learning and reflection, but may frustrate users who want direct help.
3. **Explanatory Hints.** Longer explanations that show the deduction chain (e.g., elimination reasoning, candidate interactions). These most strongly aided skill acquisition, but are costlier (longer model output) and occasionally overwhelming for novices.

We also tested mixed strategies (e.g., first a Socratic hint, then a short action hint if the user requests more). This worked much better for users unfamiliar with sudoku, but people with experience may not find the first hint useful.

3.4 Reasons for Abandonment

Several approaches were evaluated and ultimately discarded for practical reasons:

- **Pure-LLM Solver:** Allowing the LLM to both select moves and verify legality produced unacceptable rates of illegal moves and hallucinations.
- **Verbose Chain-of-Thought by Default:** Requiring models to always output full chain-of-thought was informative but produced excessive token costs and occasionally leaked internal heuristics that could confuse users. We therefore moved to concise, user-facing explanations while retaining an internal, detailed trace for developer diagnostics.
- **On-device Large Models:** Running large models locally on client devices was considered for privacy, but hardware variability and latency made it impractical for a broad user base.

These failed ideas informed the final hybrid architecture and the pragmatic choices around prompt design and hint presentation.

4 Performance Evaluation

Our evaluation objectives focused on two distinct but related measures: **move correctness** (does the suggested move preserve Sudoku constraints and progress toward solution?) and **explanation fidelity** (does the verbal rationale correspond to valid logical steps?).

Dataset and protocol. We evaluated the system on a test set composed of puzzles across three difficulty strata (Easy, Medium, Hard). For each puzzle we recorded the system's hint and verified legality with the python puzzle solution. Where applicable, we the students judged explanation accuracy on a three-point scale (correct, partially correct, incorrect).

Results (early prototype testing). During our initial hands-on testing, we ran a small number of puzzles through the system to get a rough sense of performance rather than formal metrics. Based on our observations:

- **Legality:** We estimate that almost all suggested moves were legal on puzzles up through medium difficulty, with very few rule violations (row/column/block conflicts). However, once we reached Hard difficulty levels, correctness dropped sharply — accuracy appeared closer to 50%, suggesting that deeper multi-step reasoning was required and the model struggled without the solved board as reference.
- **Move Usefulness:** Informally evaluating the suggestions as we played, we found that most (around 90%) were genuinely helpful next steps. Usefulness was highest on Easy/Medium boards. Hard puzzles still require deeper reasoning that the system occasionally misses.
- **Explanation Quality:** Explanations appeared to be mostly correct to us—somewhere in the 80–90% range by our own judgment. The biggest shortcomings came from missing intermediate logic steps or when backtracking was required to solve the cell rather than outright incorrect reasoning.

These results indicate that the hybrid design delivers high practical accuracy while preserving explanation quality. Where the model's explanation omitted intermediate reasoning, we changed how the system chooses which options to consider before it generates explanations, and because of this, the explanations became more accurate and faithful to the model's reasoning.

5 Contributions

Gillespie - Led development of the full application stack, building both the frontend interface and backend architecture. Implemented the FastAPI server, model integration, request-routing, puzzle generation, hint delivery pipeline, analytics endpoints, and interactive UI using HTML, CSS, and JavaScript. Integrated the AI tutor into gameplay, managed real-time board updates, implemented error handling, and ensured a smooth, responsive user experience across the system. Additionally deployed the project publicly and configured DNS, setting up the domain `sudokusage.com` so the application is accessible to external users.

Serov - Designed and refined the AI prompting strategy, wrote major sections of the report including methodology and development process documentation, and performed extensive quality assurance testing to evaluate model behavior and resolve inconsistencies in hint output. Contributed to slide design and presentation structure, assisted with prompt iteration and formatting decisions, and helped shape the overall reasoning style and instructional tone of the system.

Phelps - Implemented the SQLite database system for storing puzzle states and AI tutoring history. The system initializes automatically at server startup and maintains tables for puzzles and tutoring steps. Each puzzle is stored with its initial board, and every hint or correction generated by the AI is recorded with metadata such as cell location, suggested value, explanation, method used, and board states before and after the move. This database layer is designed with clean helper functions for seamless integration, supporting future features like replaying tutoring sessions, user improvement analytics, and strategy comparison.