SpindleAI Application

Michael Shomsky 3/2025

MVP - 3 Agents + Virtual Tool Manager

3 Agents

- **Solver Agent** - Uses math tools and LLM to solve
- **CAS Agent** - Uses LLM to frame the problem to be solved by a Computer Algebra System
- **Verification Agent** - Uses LLM to solve

Tools

- **MathToolbox (Python)** sum, product, divide, subtract, power, sqrt, modulo, round_number
- **MathToolbox (Rust)** calculate_average
- **Virtual Tool Manager** - Manage functions

Model

- gpt-4o-mini
- temperature = 0

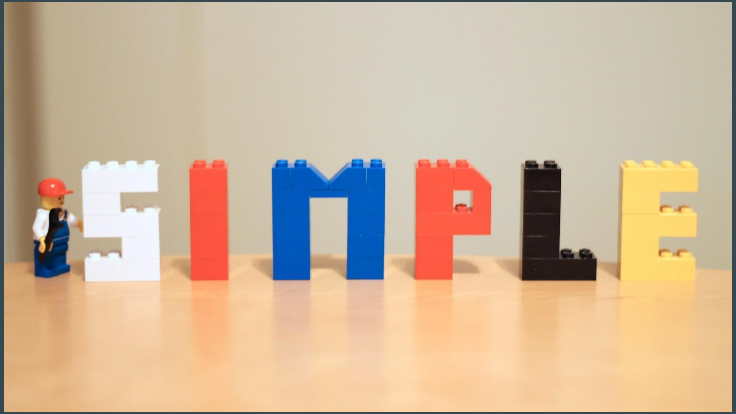
Agent Framework

- Langchain

Languages

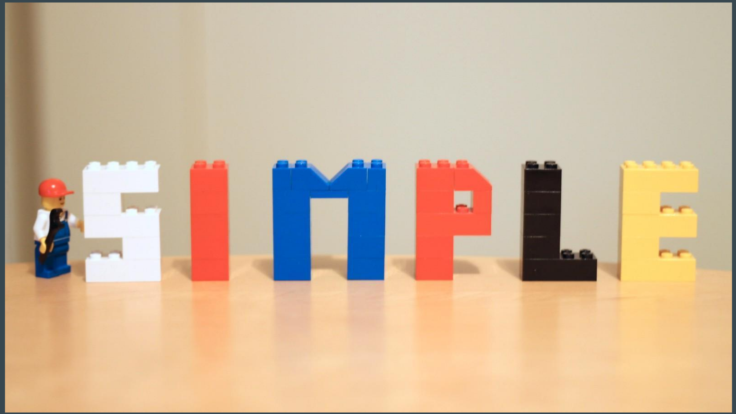
- Python + Streamlit
- Rust

MVP - Keep it Simple



- Start with a simple 3 agents
  - Solver Agent
    - Uses MathToolbox's set of math functions
    - Uses LLM to guide an answer using MathToolbox
  - Verification Agent
    - Uses LLM to guide an thorough answer
  - CAS Solution Agent
    - Uses LLM to rephrase into a question consumable by a CAS
    - Uses a CAS to form an answer

MVP - Majority Rules



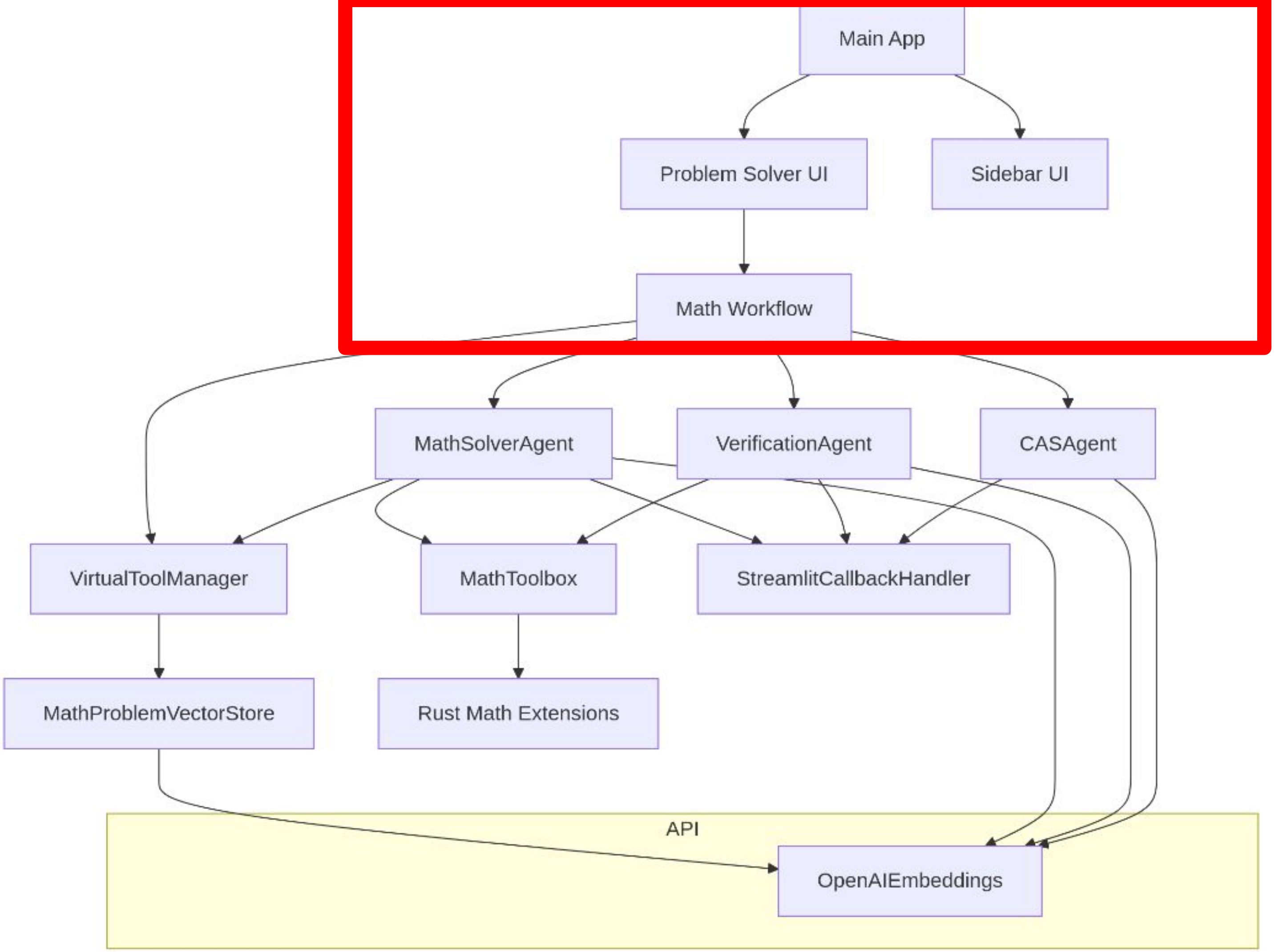
- 1. If any two or three agents agree on a solution, that solution is chosen as the majority solution.
- 2. If there's no majority agreement, we check two specific cases:

  - If the solver and CAS agents agree (but validation disagrees), we use their solution.

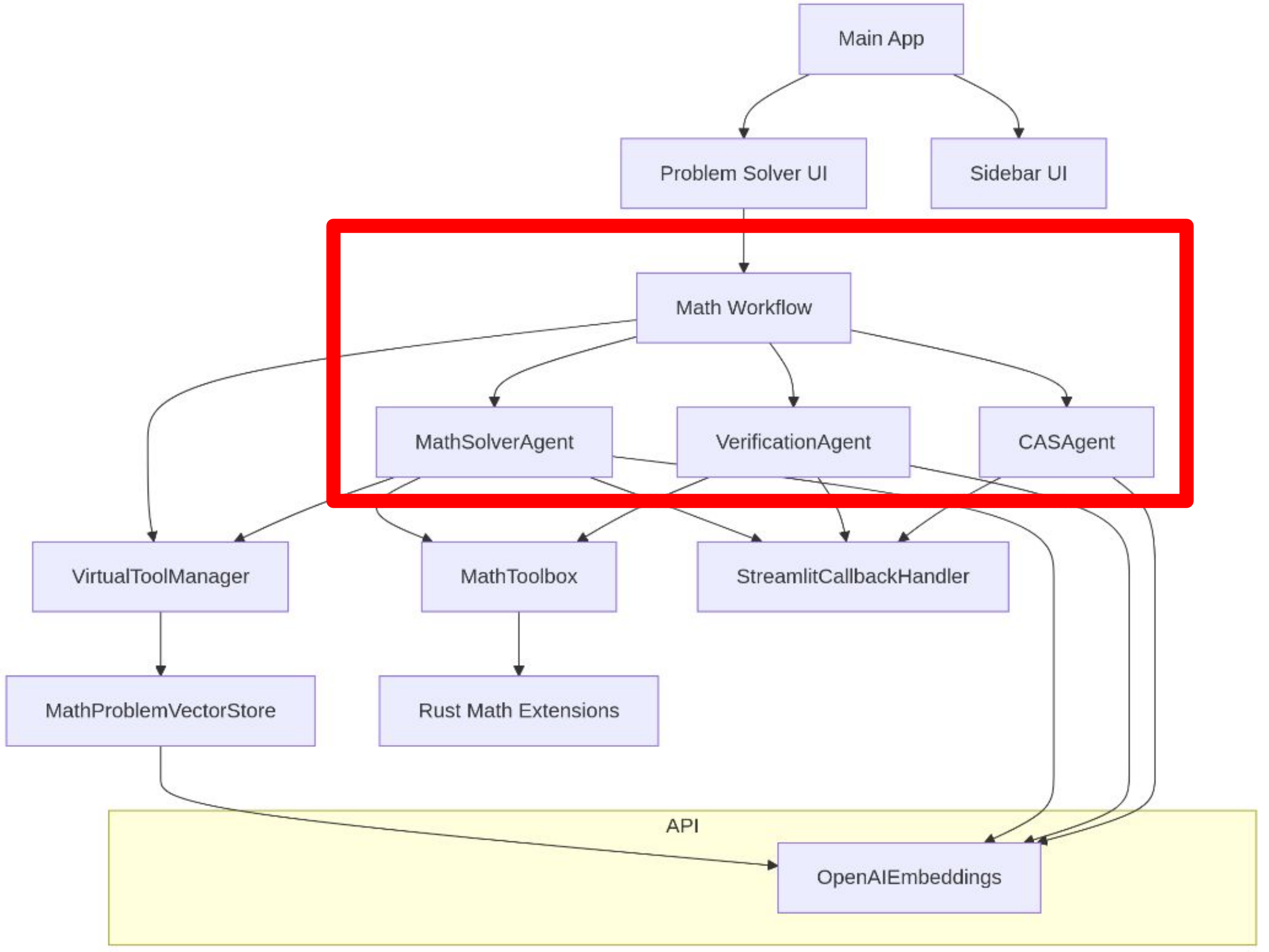
  - If the CAS and validation agents agree (but solver disagrees), we use their solution.

- 3. Only if we don't have any of the above cases, we fall back to the original behavior of using the solver solution with verification.

UI Drives a Math Workflow

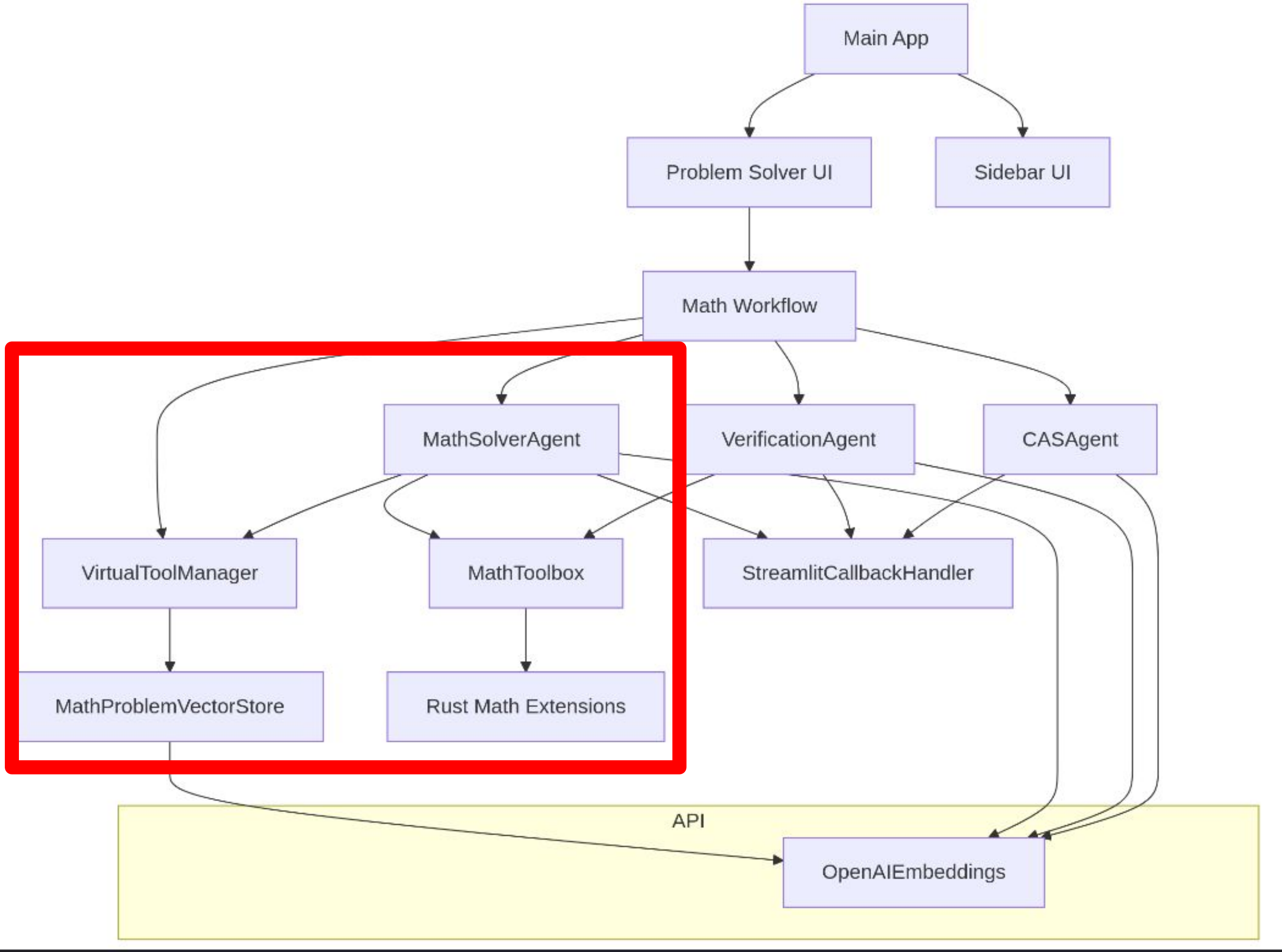


Math Workflow Orchestrates 3 Agents



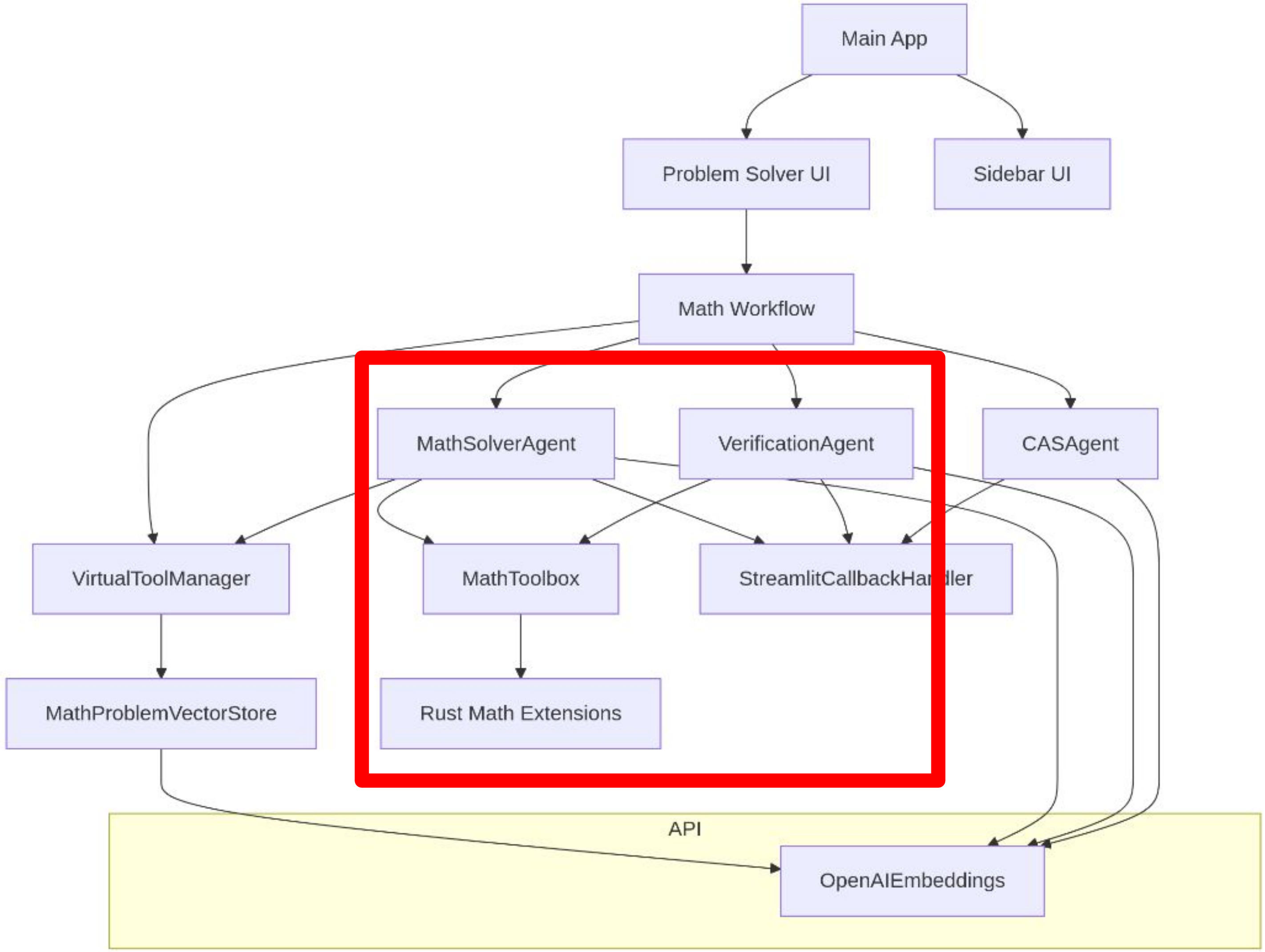
MathSolver Agents

uses Virtual Tool Manager to get relevant function



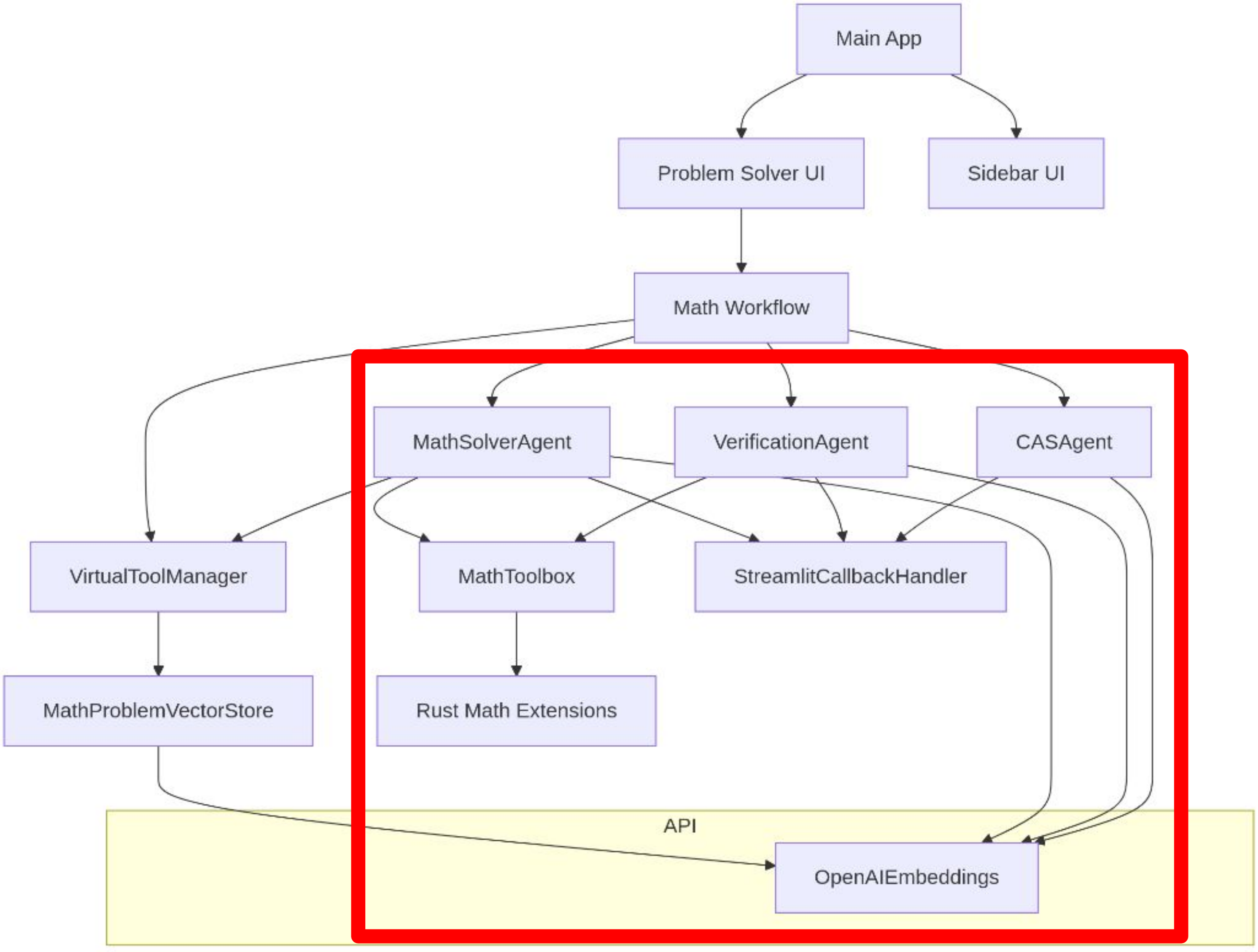
2 Agents

Rely on MathToolbox For Math Functions



3 Agents

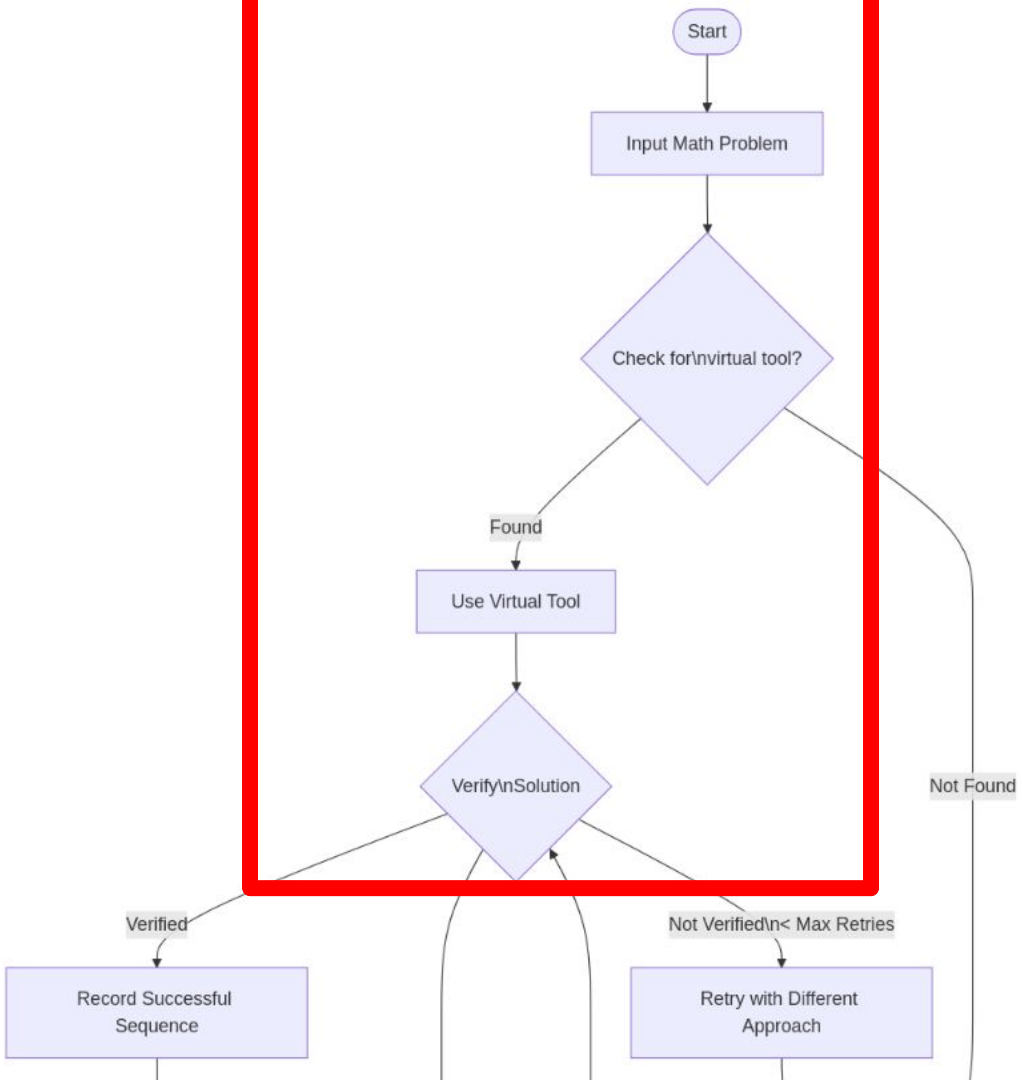
Rely on LLM integration to work with the math question



- Start by checking if there's a virtual tool.

- Use it if there is one

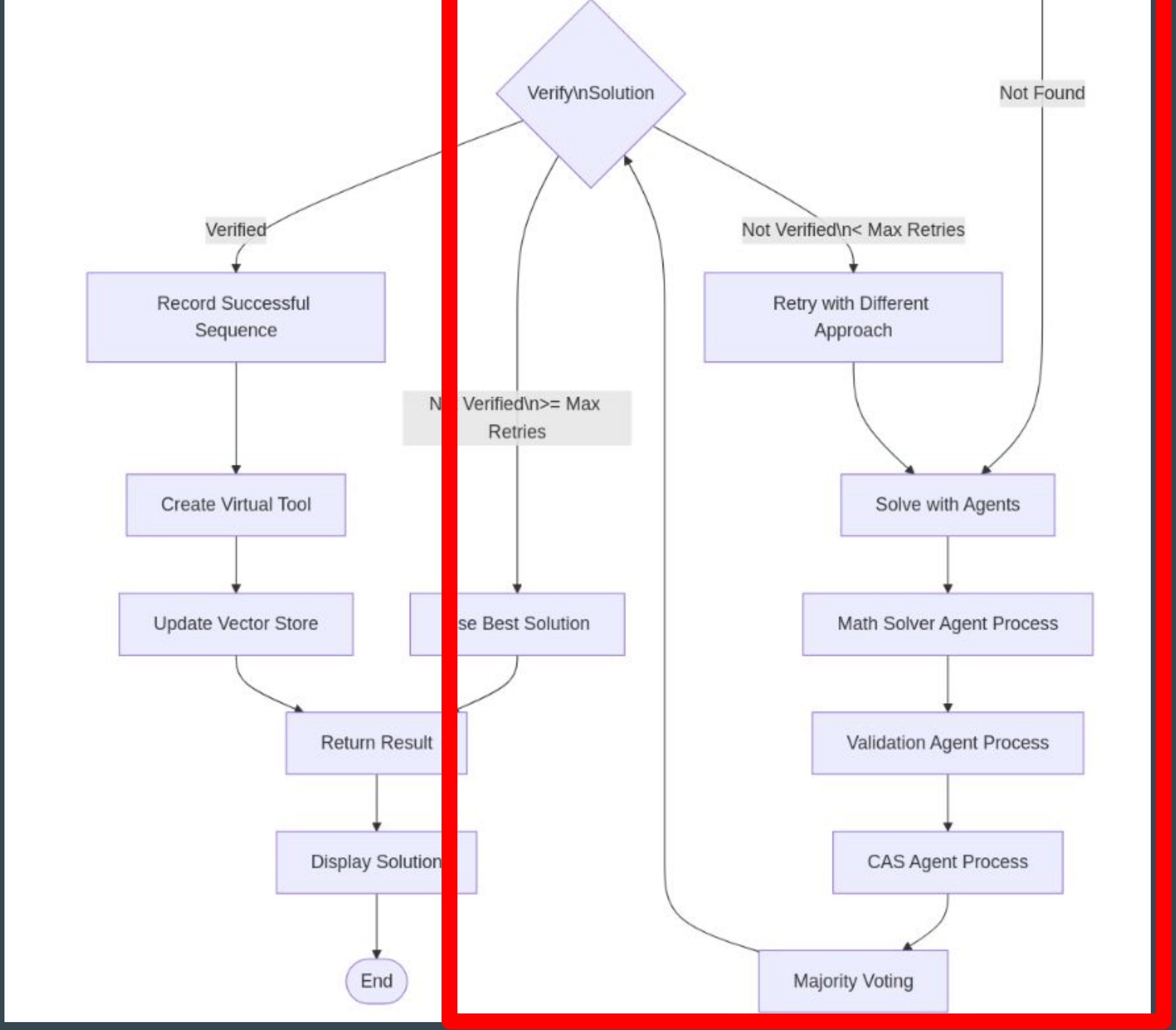
- Verify the Solution



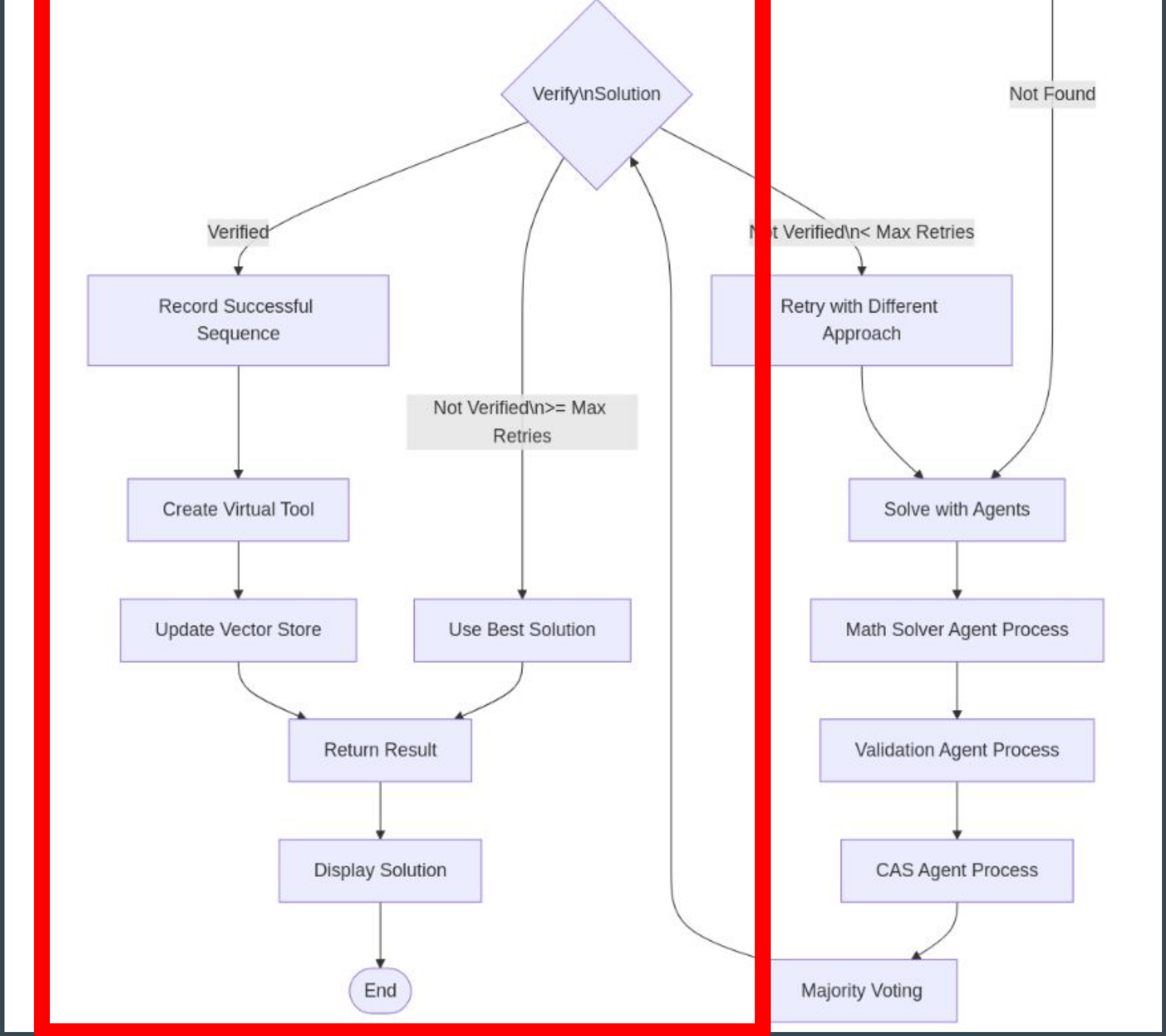
- If verification fails

- Solve with Agents

- Use Majority Voting to verify answer



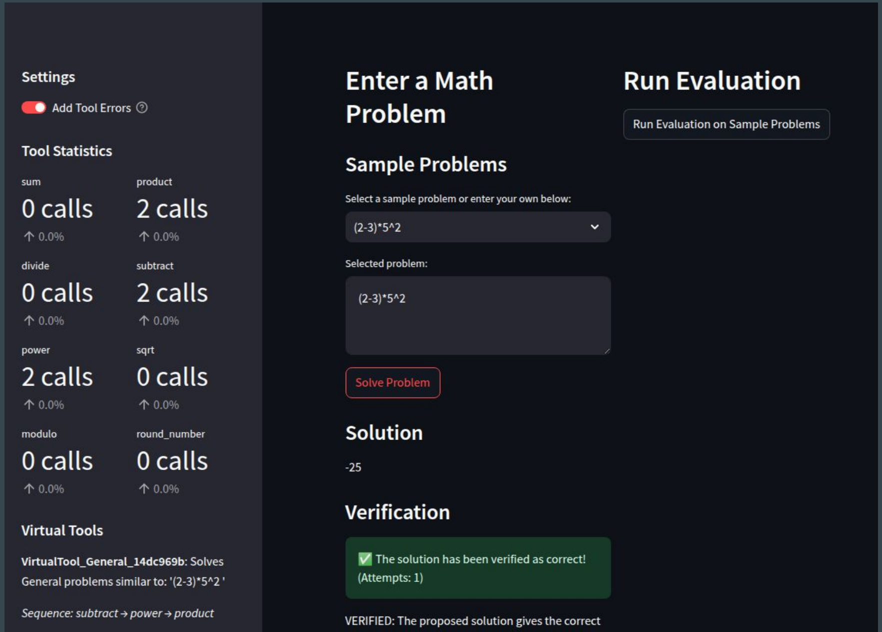
- Record successful Sequence
- Create a new virtual tool
- Update Vector Store
- Return/Display result



Demo - Python

Demo the python app using streamlit

- Live: https://mvpmathsolver.replit.app/
- Video: https://youtu.be/Z3t02ZOIFfs



Github

Invited https://github.com/orgs/spindle-app/people

https://github.com/mtshomskyieee/mvp_math_solver

- Python
- Rust (ported from Python)

Thanks

- Next Steps
  - Scaleable API using FastAPI + Kubernetes
  - Port the demo to Rust

Appendix: Requirements

https://spindle.notion.site/Coding-Project-Option-A-for-15757291437d804b87edf816a4212cdb

This mission represents a stripped-down but realistic "toy version" of the kind of multi-agent system Spindle AI is engineering (including some actual challenges we've already faced):

1. **The Setup:** First, create ≥5 distinct, simple, deterministic tools that an LLM-based agent could call to help solve user-provided math problems (*e.g.* `SUM`, `DELTA`, `PRODUCT`, `QUOTIENT`, `MODULO`, `POWER`, `ABS`, `LOG`, `TRIG`, `SQRT`, `AVG`, `MODE`, `ROUND`, `UNION`, `INTERSECT`, `DIFFERENTIATE`, `INTEGRATE`, `FACTORIZE`, … — *the specific tools are entirely up to you*).
    1. **Modify 1-2 of the most basic tools to *intentionally but silently throw errors (and/or silently give incorrect answers) 30%-50% of the time the tool is called*.** You *may* also want to include a basic `GET_USER_INPUT` tool for requesting input/clarification from a human user. (You can organize all tools in some form of "toolbox" if you want, but we'd prefer you do **not** hardcode a string listing all the tools, their docs, and their usage examples in a *single* prompt file or prompt mega-string anywhere in the project.)
2. **The Architecture:** Prototype a multi-agent system with ***at least* 2 agents** and *at most* 5 agents (for whatever definition of "agent" you believe makes sense in this context), that discovers which tools are available and sequences tool calls to **reliably** solve basic user-provided math problems (or if you prefer, mathy word problems). The agents can *only* ****use the available tools ***(including the unreliable tool[s])***, *i.e.* no LLM-hallucinated arithmetic should be used for user-facing answers (*even* if that arithmetic is correct, as is increasingly the case among frontier models).
    1. You might well choose to include a lightweight planning, reasoning, and/or task decomposition layer in your prototype — but unless you have a compelling justification, all *user-facing* outputs (and most intermediate outputs) should be structured or semistructured, not unstructured.
    2. **Don't hesitate to ask us for an OpenAI API key or Anthropic API key.** Otherwise, we're happy to reimburse these costs after submission *(within reason/at Spindle's discretion)*.
3. **The Twist:** When your prototype identifies a sequence of tool calls that reliably *or fairly reliably* solves a certain class of math problem(s) **based on successful execution(s)**, it should learn to do something like *(e.g.)* **memoize or semantically cache that sequence of tool calls as a single, idempotent new** `VirtualTool` (*i.e.* some learning behavior akin to **"bundling" the tool calls into a *single* new idempotent tool, to which a *single* call can be made, which can be reliably invoked *next time a math problem of the same or similar form is encountered*).
4. **The Finish Line:** Prove programmatically that your prototype works reasonably well (or at least that it could be *completed* to work reasonably well, if short on time).
    1. **Bonus points for using actual evals to show this.**
        1. ***(If you're are an "evals-focused" candidate, consider reframing/approaching the entire task through the lens of an evals system instead, i.e. evals-driven development. Just tell us to judge your quality vs. emphasis vs. completion accordingly.)***
5. **Bonus Points:**
    1. Create the math toolbox/interfaces in a non-Python language (ideally Rust, Go, or Typescript).
    2. *If* you decide to use a vector database anywhere, consider prototyping your *own* vector DB or VDB-like utility. (Not if this takes up all your time, though. It's not the most important part.)

- **If you don't have enough time for a project like this, or have alternate ideas, please let us know so we can find a path forward that we all feel good about!** Either way, we really look forward to seeing you through these next steps.