AI Development Log

CollabBoard — Real-Time Collaborative Whiteboard with AI Agents

---

Tools & Workflow

CollabBoard's AI features are powered by two separate systems: an **AI Command Bar** (LangChain + OpenAI GPT-4o-mini) for natural-language board actions, and a **Multi-Agent Visual Collaboration** system (Anthropic Claude Haiku 4.5) where AI personalities collaborate on the canvas in real time. Both use **Zod** schema validation and **Langfuse** observability.

The Command Bar follows a structured-output pattern: user prompts and serialized board state go to GPT-4o-mini in JSON mode, returning an array of typed board actions validated against a Zod discriminated union of 13 action schemas, then executed client-side against the tldraw editor API.

The Multi-Agent system sends prompts to Claude Haiku 4.5 with personality-specific system prompts. Each agent returns structured actions that are executed with animated cursor simulation — agents appear as real users with avatars gliding across the canvas as they place stickies, shapes, frames, and arrows.

Multiple AI coding tools were used throughout development: **Claude Code** for scaffolding, refactoring, and test generation; **GitHub Copilot** for inline completions; and **ChatGPT** for code reviews, requirements analysis, and comparing architectural approaches.

MCP Usage

No external MCP servers were used at runtime. During development, Claude Code with its built-in tools (file read/write, bash, grep, glob) was used for agentic coding tasks — scaffolding the agent architecture, writing tests, and debugging integration issues. ChatGPT was used conversationally for requirements review, brainstorming feature approaches, and comparing technology trade-offs. The Langfuse integration acts as the observability layer that an MCP-based tracing server would typically provide.

Effective Prompts

1. **System prompt — tool schema enforcement**: "Every action object in the 'actions' array MUST have a 'type' field set to one of these EXACT strings: create_sticky, create_multiple_stickies, create_text, create_shape..." — Eliminated type invention by GPT-4o-mini.

2. **Board-state context injection**: "CURRENT BOARD STATE (N items): [{id, type, x, y, text}...]" — Appended to every user message so the LLM can reason about existing content, avoid overlaps, and reference shape IDs.

3. **Layout guidelines in system prompt**: "Sticky notes are roughly 200x200 pixels. Space them at least 230px apart. For brainstorming, use a grid layout starting from (-400, -300)." — Produced clean, readable layouts without post-processing.

4. **Template examples**: "SWOT Analysis: Create 4 frames in a 2x2 grid, each ~500x400. Add starter stickies inside each." — One-shot examples for common templates dramatically improved output quality.

5. **Type normalization fallback**: A 25+ entry alias map (e.g., createSticky to create_sticky, note to create_sticky) catches LLM type-name drift, improving reliability from ~85% to ~99% parse success.

6. **Color normalization (Multi-Agent)**: A 15+ entry color alias map (e.g., pink to light-red, purple to violet) normalizes LLM-invented colors to valid tldraw palette values before Zod validation.

Code Analysis

Approximately **95% AI-generated, 5% hand-written** (across Claude Code, Copilot, and ChatGPT combined). Claude Code generated the initial scaffolding for both the Command Bar agent and Multi-Agent system, all Zod schemas, and test suites. Copilot assisted with inline completions. ChatGPT was used for reviewing requirements against the spec and comparing architectural options.

Hand-written work focused on the type normalization map, Langfuse integration tuning, Yjs-tldraw sync logic in useYjsStore.ts, the PartyKit server, agent cursor simulation timing, and production hardening (error boundaries, timeout handling, cleanup refs).

Strengths & Limitations

Where AI excelled: Generating Zod schemas for all 13 action types, writing comprehensive test suites (1,600+ lines), scaffolding the LangChain and Anthropic SDK integrations, iterating on system prompt wording, building the agent cursor simulation system, and creating the external bot invite UI with API key auth.

Where it struggled: Getting tldraw-specific APIs right (shape creation props, editor method signatures) required manual correction — particularly strict prop validation where extra properties like 'align' or 'font' cause runtime crashes. The Yjs-tldraw sync bridge had subtle race conditions that AI couldn't debug without human guidance. Server-side Yjs connections to PartyKit failed silently, requiring an architecture pivot to client-side execution. Firebase credential configuration and environment-specific issues required domain knowledge.

Key Learnings

Structured output (JSON mode + Zod validation) is the single most impactful pattern for AI agents — it eliminates free-form parsing and makes LLM output deterministic enough for programmatic consumption. A type-alias normalization layer is essential because even with explicit instructions, LLMs occasionally invent type names. Similarly, color normalization is critical when LLMs generate visual content — they frequently use color names outside the target palette.

Client-side execution proved more reliable than server-side Yjs writes for agent actions. One-shot examples in system prompts are worth more than paragraphs of instructions. Langfuse tracing is invaluable for debugging latency spikes and cost attribution. Finally, animated cursor simulation transforms AI from 'magic output appearing' to 'watching a collaborator think and work' — a much more engaging user experience.

AI Cost Analysis

Development & Testing Costs

Metric	Value
LLM API costs (OpenAI)	~$0.80
LLM API costs (Anthropic)	~$0.15
Input tokens consumed	~380,000
Output tokens consumed	~105,000
Total API calls	~180
Langfuse hosting	Free tier (50K observations/mo)
Other AI costs	$0 (no embeddings, no fine-tuning)

Production Cost Projections

Assumptions: 5 AI commands per user per session, 8 sessions per user per month. Command Bar: ~1,200 input / ~500 output tokens per command (GPT-4o-mini: $0.15/1M in, $0.60/1M out). Multi-Agent: ~2,000 input / ~800 output tokens per invocation (Claude Haiku 4.5: $0.80/1M in, $4.00/1M out).

Scale	Commands/mo	Input Tokens	Output Tokens	Est. Monthly Cost
100 users	4,000	~6M	~3M	~$4.50
1,000 users	40,000	~60M	~30M	~$45
10,000 users	400,000	~600M	~300M	~$450
100,000 users	4,000,000	~6B	~3B	~$4,500

At scale, costs are dominated by Claude Haiku output tokens for the Multi-Agent system. Caching system prompts could reduce input costs by ~50%. The Command Bar (GPT-4o-mini) remains extremely cost-effective at all scales.