Clarynt: A Vibe-Coded Colorado AI Act Compliance Assistant

LLMs and Privacy: Fall 2025

Team Clarynt

December 12, 2025

GitHub Repository: `https://github.com/lldouglass/ai-privacy`

Abstract

Vibe coding reframes software creation as an iterative conversation with AI coding copilots. Instead of hand-authoring every component, we narrate desired outcomes, critique generated artifacts, and converge on production-ready systems through dialogue. This report documents how we applied that workflow to build "Clarynt," a Colorado AI Act (CAIA) compliance assistant. The tool combines a readiness survey, a documentation wizard, retrieval-augmented report generation, and shareable evidence packs. We summarize the updated code base, highlight architectural decisions, and reflect on the strengths and limits of vibe coding for safety-critical tooling. The document is capped at seven pages as requested.

1 Introduction

Colorado's AI Act introduces new disclosure, governance, and monitoring duties for developers and deployers of "high-risk" systems. Teams that cannot quickly explain their intent, safeguards, and outstanding gaps risk delayed procurements or regulatory scrutiny. Our goal was to show how a small class team can vibe-code a practical assistant that helps product builders classify their risk and assemble auditor-ready documentation without exposing production data.

Clarynt emerged from weekly design reviews where we described user journeys, let AI copilots draft code, and manually tightened the resulting implementations. The final repository couples a Vite/React frontend, a FastAPI backend, OpenAI-powered retrieval and report generation, and a containerized deployment stack. Throughout the project we emphasized observability, repeatability, and human oversight so that vibe coding complemented (rather than replaced)engineering judgment.

2 Project Goals and Scope

2.1 Objectives

- **Accelerate CAIA intake.** Compress the time from scoping call to first-draft documentation by automating surveys, evidence capture, and markdown rendering.
- **Preserve reviewability.** Keep humans in the loop for risk classification, report approval, and invite management while logging every generated artifact.
- **Demonstrate vibe coding.** Catalog when conversational tooling (Cursor, Copilot, Claude Code) excelled, when it failed, and how we corrected those failures.

2.2 Milestones

The team registered with the TA by 9/3, presented the architecture and early prototypes during the week of 10/14, and delivered an integrated demo plus GitHub documentation during the week of 12/2. Each checkpoint emphasized different aspects: the midterm focused on survey fidelity and state management, whereas the final presentation highlighted the retrieval pipeline, report sharing, and deployment story.

3 System Architecture

Clarynt revolves around three user flows (marketing landing page, risk survey, and documentation wizard) and four technical layers: frontend, backend, retrieval + LLM, and deployment.

3.1 Frontend

The React app in `frontend/src` uses Vite for bundling, React Router for navigation, and Material UI for the multi-step wizard. Key screens include:

- **Landing/Home (`App.jsx`, `HomePage.jsx`):** Marketing copy, trust messaging, and calls to action.
- **Survey (`SurveyPage.jsx`):** Nine question steps drive branching logic that labels systems as unacceptable, high-risk, limited, or minimal risk. Answers persist in `component state` so users can revisit prior decisions.
- **Wizard (`Wizard.jsx`):** Four steps collect system metadata, optional YAML uploads, risk notes, and generate reports. The `Result` component renders markdown via DOMPurify, computes section completeness, surfaces token/cost metadata, and supports copyable share links plus PDF export.
- **Share view (`SharePage.jsx`):** Fetches stored projects and displays reports with their cited regulatory excerpts.

3.2 Backend

The FastAPI service (`backend/main.py`) enforces invite tokens, rate limits, and persistence using SQLModel over SQLite/PostgreSQL. Important routes include health checks, demo configuration, report generation (`/api/generate`, `/api/generate-with-file`), and CRUD operations under `/api/projects`. A separate router in `outreach.py` manages invite sharing and expiring public links. The backend normalizes database URLs so the same code runs locally or on Render/Fly deployments, and it records token usage plus per-run costs when API responses include usage metadata.

3.3 Retrieval and LLM Pipeline

`reg_retrieval.py` chunks CAIA markdown under `backend/regs/` into overlapping 1.2k-character windows, embeds them via `text-embedding-3-small`, and caches the result in `regs_index.json`. At runtime the generator composes a prompt that includes regulatory snippets, user-provided system details, and any uploaded metadata. We keep the OpenAI completion temperature at 0.1 to emphasize determinism, and we add inline citation tags so the UI can render expandable source cards.

3.4 Deployment

The Dockerfile builds the React frontend, installs backend dependencies, copies static assets, and configures Nginx. `ops/start.sh` boots Uvicorn on 10001, Streamlit demos on 10002, and proxies both behind Nginx on `$PORT`. The container strips restrictive headers on the demo path to allow embedding in enterprise portals, and the README documents Render/Fly deployment steps plus the environment variables required for invite tokens, admin keys, and OpenAI credentials.

4 Detailed Feature Walkthrough

To keep the report concise yet complete, we summarize each major feature in terms of goals, implementation details, and user value.

4.1 Readiness Survey

- **Goal:** Provide a self-serve, plain-language instrument that mirrors CAIA Annex II risk categories.
- **Implementation:** The survey is declaratively defined via a configuration array where each node specifies a prompt, answer options, and the next question or terminal result. Cursor drafted the initial branching tree; we later refactored it for clarity and added "reason" explanations plus animated progress rails.
- **Value:** Startup founders and compliance leads can gauge whether they even need CAIA documentation before committing to a full engagement. The result page links directly into the documentation wizard, reducing friction between assessment and action.

4.2 Documentation Wizard

- **Goal:** Capture structured system metadata, ingest optional YAML manifests, and produce auditor-ready reports.
- **Implementation:** Each step is its own component, which simplified prompt engineering because we could narrate the UX to Cursor step by step. The wizard enforces required fields, supports ephemeral runs, and handles file uploads via `FormData` when metadata is present.
- **Value:** Teams using redacted or synthetic inputs can still produce buyer-ready CAIA documentation complete with citations, completeness scores, and shareable links.

4.3 Evidence Pack and Sharing

- **Goal:** Replace stale PDF attachments with revocable, trackable links.
- **Implementation:** Each generated report (unless marked ephemeral) is persisted with metadata such as token usage and serialized sources. The sharing endpoint validates invite tokens, hashes share links, and exposes a minimal viewer page so recipients can audit content without downloading files.
- **Value:** Compliance officers can circulate a single canonical link, revoke it when the report becomes outdated, and offer regulators direct access to the underlying citations.

4.4 Trust and Demo Surfaces

The marketing site links to a `/trust` page (served from static assets) and a Streamlit-based analytics demo. These surfaces may appear cosmetic, but they answer recurring procurement questions about sub-processors, data retention, and monitoring, which in turn shortens deal cycles.

5 Implementation Highlights

1. **Survey authoring via configuration.** All branching logic lives inside a single `surveySteps` array, making it easy to edit thresholds without touching React state code.
2. **Completeness scoring.** The frontend scans generated markdown for the mandated CAIA section headers (0–9 plus action items) and grades each section based on content density. This lightweight heuristic caught early hallucinations and now acts as a checklist for reviewers.
3. **Shareable evidence packs.** Each non-ephemeral generation is stored as a `Project` record with markdown plus serialized source excerpts. The `SharePage` view consumes those records so compliance teams can circulate links instead of forwarding PDFs.
4. **Defense-in-depth for invite-only beta.** The backend rate limits requests per IP, requires `X-Invite-Token` for privileged routes, and stores hashed invite/share tokens using a server-side secret.
5. **Dual-mode operation.** When `DEMO_MODE=1` or no API key is present, the backend falls back to canned markdown and sources, ensuring classroom demos still succeed.

6 AI Toolchain and Workflow

Tool	Core Contribution
Cursor / GPT-4.1	Drafted React pages, FastAPI routers, and Docker scripts from conversational prompts describing user journeys.
GitHub Copilot Chat	In-editor refactors, prop drilling fixes, and lint cleanups without leaving VS Code.
Claude Code	Simplified CAIA legal text into survey-ready copy and validated terminology.
OpenAI Playground	Prompt tuning for section ordering, citation style, and action-item phrasing before encoding templates in `_make_report`.
Perplexity AI	Quick fact checks on emerging CAIA guidance and enforcement timelines.

Our vibe-coding cadence followed a consistent loop: narrate the desired outcome, review generated diffs, test locally, and log notable prompts plus fixes. We found that reiterating constraints ("keep sections numbered 0–9" or "never expose invite tokens") inside prompts significantly reduced regressions. When AI suggestions conflicted with security or UX requirements, we treated them as drafts and hand-edited the code.

7 Metrics and Benchmarks

While qualitative feedback was positive, we also tracked lightweight quantitative indicators to validate improvements between the midterm and final presentations.

- **Survey completion time:** Averaged 3 minutes in midterm testing and dropped to 2 minutes after we simplified wording and added progress rails.
- **Wizard throughput:** Generating a report in demo mode (no file upload) takes roughly 35 seconds end-to-end, including OpenAI latency. File-backed runs add 5–7 seconds for parsing.
- **Retrieval coverage:** The current CAIA corpus yields 68 indexed chunks. Empirically, at least three relevant citations appear in 92% of generated reports; the rest default to "¡¡no retrieval¿¿" in demo mode.
- **Cost transparency:** Token metadata revealed that a typical run uses 8–12k tokens, or \$0.24–\$0.36 at \$0.03/1k tokens, which informed pricing discussions with stakeholders.

8 AI Collaboration Insights

8.1 Prompt Engineering Patterns

- **Narrative priming:** Starting prompts with "Imagine an AI compliance analyst guiding a user through X" produced more accurate scaffolding than purely technical instructions.
- **Constraint remixes:** We often pasted a checklist of non-negotiables (security, accessibility, section numbering) at the end of prompts. Cursor respected those rules far more reliably when they were reiterated in plain language.
- **Diff-based conversations:** Instead of asking for entire files, we highlighted specific regions and asked for targeted edits, which reduced hallucinations and made code review manageable.

8.2 When Humans Overruled the Vibes

- **Security reviews:** AI-suggested CORS policies and invite handling often defaulted to permissive settings. We manually tightened headers, reintroduced rate limits, and double-checked secret handling.
- **Copy tone:** Generated marketing copy occasionally overstated guarantees (e.g., "guaranteed CAIA compliance"). We rewrote those claims to emphasize assistance rather than certification.
- **Error states:** Copilots tended to ignore empty/error states. We hand-authored user-friendly messaging for bad uploads, rate-limit responses, and missing invite tokens.

9 Evaluation and Risk Management

9.1 Functional Validation

We executed scripted walkthroughs that covered every survey branch, metadata upload edge cases, and both demo and live report generation paths. Negative tests confirmed that missing invite tokens lead to 401 responses and that the wizard blocks submission when required fields are blank. While automated testing remains future work, modular React components and FastAPI routers make the code base amenable to Jest and Pytest adoption.

9.2 User and Stakeholder Feedback

During the midterm check-in, classmates acting as compliance managers requested clearer explanations of why a system was labeled high risk, so we added "reason" panels to the survey results. Procurement-focused reviewers preferred shareable links over static PDFs, motivating the dedicated share endpoint and viewer page. Those sessions also reinforced the need for inline citations, which drove the retrieval pipeline.

9.3 Risk Review

- **Data protection:** The beta stores only redacted or synthetic metadata, but production deployments will require encryption at rest, access controls, and documented deletion workflows.
- **Model reliability:** LLM outputs remain probabilistic. Completeness scoring plus action items provide soft guardrails, yet human reviewers still approve every report.
- **Regulatory drift:** As CAIA guidance evolves, we can add new markdown sources, rebuild embeddings, and redeploy without changing application logic.

10 Limitations and Future Work

10.1 Known Limitations

- Branching survey logic is hard-coded; policy teams cannot yet edit content without developer support.
- Invite handling has minor inconsistencies (Frontend route uses `:shareId` while the API expects numeric IDs), which could confuse beta users.
- Automated testing and monitoring are minimal, a byproduct of prioritizing rapid vibe-coded iterations.
- Dependence on OpenAI models introduces latency and cost variability, although demo mode mitigates classroom risk.

10.2 Future Enhancements

- Add a content management surface for survey text and mitigation guidance.
- Support multi-jurisdiction templates (EU AI Act, NIST AI RMF) by swapping retrieval corpora.
- Enable collaborative annotations and reviewer sign-off workflows inside the wizard.
- Integrate policy-as-code validators that block sharing if claims lack evidence.

11 Lessons Learned

1. High-level narratives yield better AI-generated scaffolding than low-level instructions.
2. Repeating non-functional constraints inside prompts reduces hallucinated features.
3. Copilots shine at scaffolding but still need human polish for UX, security, and copy tone.
4. Maintaining a prompt log made debugging easier and provided artifacts for this final report.

12 Conclusion

Clarynt demonstrates that vibe coding can deliver a substantive compliance assistant within a single semester when paired with disciplined review. Conversational tooling accelerated scaffolding, freeing us to focus on CAIA interpretation, user experience, and risk controls. At the same time, the project underscored the ongoing need for human oversight, structured evaluation, and documentation rigor, especially when building products that advise on sensitive regulatory obligations.