COPILOTLENS: Layered Explanations for AI-Generated Code to Support Developer Understanding

Boushra Almazroua[1], Zeling Zhang[2], Runlong (Harry) Ye[2], Michael Liut[3]
[1]King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
[2]Department of Computer Science, University of Toronto, Toronto, Canada
[3]Mathematical & Computational Sciences, University of Toronto Mississauga, Mississauga, Canada

***Abstract*—AI coding assistants like GitHub Copilot, Cursor, and Claude are widely used but lack transparency, often producing code without explaining why it was generated. This opacity hinders student learning and professional trust. We present COPILOTLENS, a two-level explanation framework that makes AI-generated code transparent through post-hoc summaries and on-demand reasoning. A survey with 74 undergraduate students identified pain points and feature priorities; participants strongly desired *design rationale* explanations and clear visual diffs. We developed a prototype, presented it at the SUDS Showcase, and our short paper was accepted to COLM 2025. This report summarizes objectives, milestones, system design, methodology, findings, challenges, and future work.**

***Index Terms*—Explainable AI, Coding Agents, Human-Centered AI, LLMs, Developer Tools, Transparency**

I. INTRODUCTION

AI-powered coding assistants such as GitHub Copilot, Cursor, Claude, and Gemini accelerate software development by generating code suggestions and even performing multi-file edits. However, these systems often lack transparency. While generated code may function, developers are left without an understanding of *why* a solution was chosen, what parts of the codebase influenced the output, or which alternatives were considered. The result is a gap between productivity and comprehension: novices lose learning opportunities, and professionals face fragile trust in AI-generated code.

We introduce **COPILOTLENS**, an interpretability layer that reconstructs an agent's "thought process" post-hoc. Level 1 provides concise summaries of what changed and why; Level 2 offers evidence-based rationale including codebase influences, coding conventions, and alternative implementations.

Our contributions include: (1) identifying developer pain points via a 74-participant survey, (2) implementing a layered explanation prototype, (3) presenting at the SUDS Showcase, and (4) publishing our short paper, *Beyond Autocomplete: Designing COPILOTLENS Towards Transparent and Explainable AI Coding Agents*, accepted at the COLM 2025 Workshop on LLM Explainability [1].

II. PROJECT OBJECTIVES AND GOALS

The project aims to design and evaluate an explainable AI-assisted coding tool that improves learning and trust by exposing the reasoning behind AI-generated code.

Goals:

- **Identify pain points:** Document challenges such as complex outputs, deprecated dependencies, and lack of rationale.
- **Prioritize features:** Use survey data to select explanation features that best support comprehension and calibrated trust (e.g., Design Rationale, Visual Diff).
- **Develop a prototype:** Build a two-level explanation system: (i) post-hoc summaries; (ii) on-demand extended reasoning with evidence and alternatives.
- **Evaluate empirically:** Combine surveys and planned interviews to assess understanding, trust, and error detection.
- **Disseminate results:** Poster/demo, final report, and COLM 2025 workshop publication.

III. CURRENT PROGRESS AND MILESTONES ACHIEVED

Weeks 1–2 (Foundation): literature review on LLM + code; planned mixed-methods methodology (survey + interviews); drafted and submitted COLM short paper.

Weeks 3–4 (Survey & Acceptance): mapped related work to features; designed/piloted Qualtrics survey; launched study; COLM 2025 workshop acceptance.

Week 5 (Data & Prototype): collected 74 responses, cleaned/aggregated via Python scripts; identified top features (Design Rationale, Visual Code Diff); produced prototype concepts.

Showcase: presented poster and live demos at the SUDS Showcase; integrated feedback into this report and the workshop paper.

IV. SYSTEM DESIGN: COPILOTLENS

Modern agents produce multi-file changes with minimal explanation, limiting learning and trust. COPILOTLENS adds a layered interpretability layer on top of an open-source coding agent (Kilo Code).

A. Overview

As shown in Figure 1, mainstream tools (left) present only the final code artifact, whereas COPILOTLENS (right) exposes reasoning through two levels.

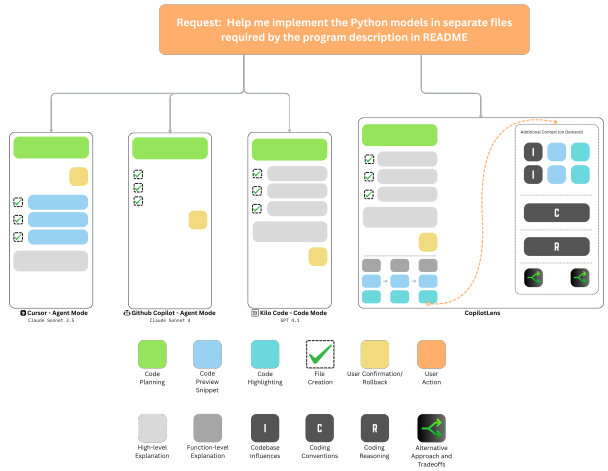


Fig. 1. Comparison: mainstream agents vs. COPILOTLENS. The latter reveals reasoning, influences, and alternatives.

Design principles:

1) **Reveal the agent's thought process.**
2) **Support informed critical evaluation.**
3) **Make reasoning tangible and verifiable.**

B. Level 1: Post-Hoc Summaries

Immediately after the agent completes a task, COPILOTLENS presents a concise summary (Figure 2):

- **File navigator** of modified files;
- **Step titles** naming each change;
- **One-sentence rationales** describing significance;
- **Inline code previews** with highlights.

This answers: *What just happened?*

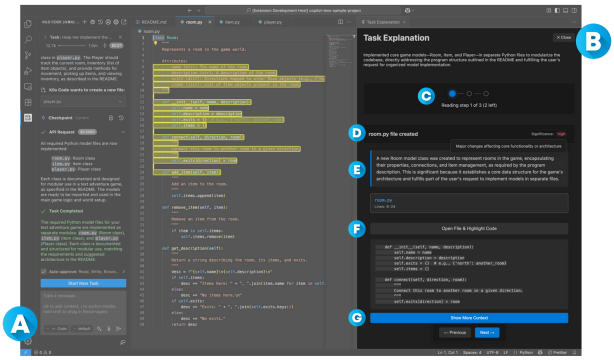


Fig. 2. Level 1: post-hoc summaries with file navigator, significance, and inline previews.

C. Level 2: On-Demand Extended Reasoning

Developers can expand to Level 2 (Figure 3) for evidence-based explanations:

- **Codebase Influences** (what files/docs guided the change);
- **Coding Conventions** (style/patterns used);
- **Implementation Reasoning** (why this approach);
- **Alternative Implementations** (other viable options with trade-offs).

This answers: *Why was it done this way?*

D. Core Features

1) **Change Summary** — high-level overview of file modifications.
2) **Visual Code Diff** — Git-style line-by-line edits.
3) **Design Rationale** — *why* a design choice was made (top-rated feature).
4) **Alternative Implementations** — options & trade-offs to prevent blind acceptance.
5) **Codebase Influences** — traceability to concrete project files.
6) **Coding Conventions** — style/idioms followed.
7) **Inline Confidence Highlighting** — uncertainty cues for triage.
8) **Dynamic UML Diagrams** — lightweight relationships across files/classes.
9) **Proactive Understanding Checker** — optional prompts to reinforce comprehension.

V. METHODOLOGY AND APPROACH

We adopted a mixed-methods strategy combining surveys, prototype development, and planned interviews.

A. Survey

A Qualtrics survey was designed to capture demographics, tool familiarity, usage contexts, perceived benefits and pain points, and evaluations of explanation features such as *Change Summary*, *Visual Diff*, and *Design Rationale*. The survey included Likert scales, ranking tasks, and open-ended prompts. We recruited 74 undergraduates in Computer Science and related fields, most in their 2nd–3rd year. After de-identification and cleaning with Python scripts, 70 valid responses remained for analysis. Descriptive statistics and thematic coding were applied, with 66 students volunteering for follow-up interviews.

B. Prototype

The prototype was built on top of the open-source Kilo Code agent. COPILOTLENS added two layers of explanations: (1) post-hoc summaries of file-level changes, and (2) on-demand reasoning showing influences, conventions, rationale, and alternatives (Figures 2–3).

C. Next Phase

The next step is to conduct semi-structured interviews with the 66 volunteers. These studies will evaluate comprehension, trust calibration, and error detection by comparing COPILOTLENS against a baseline coding agent without explanations.

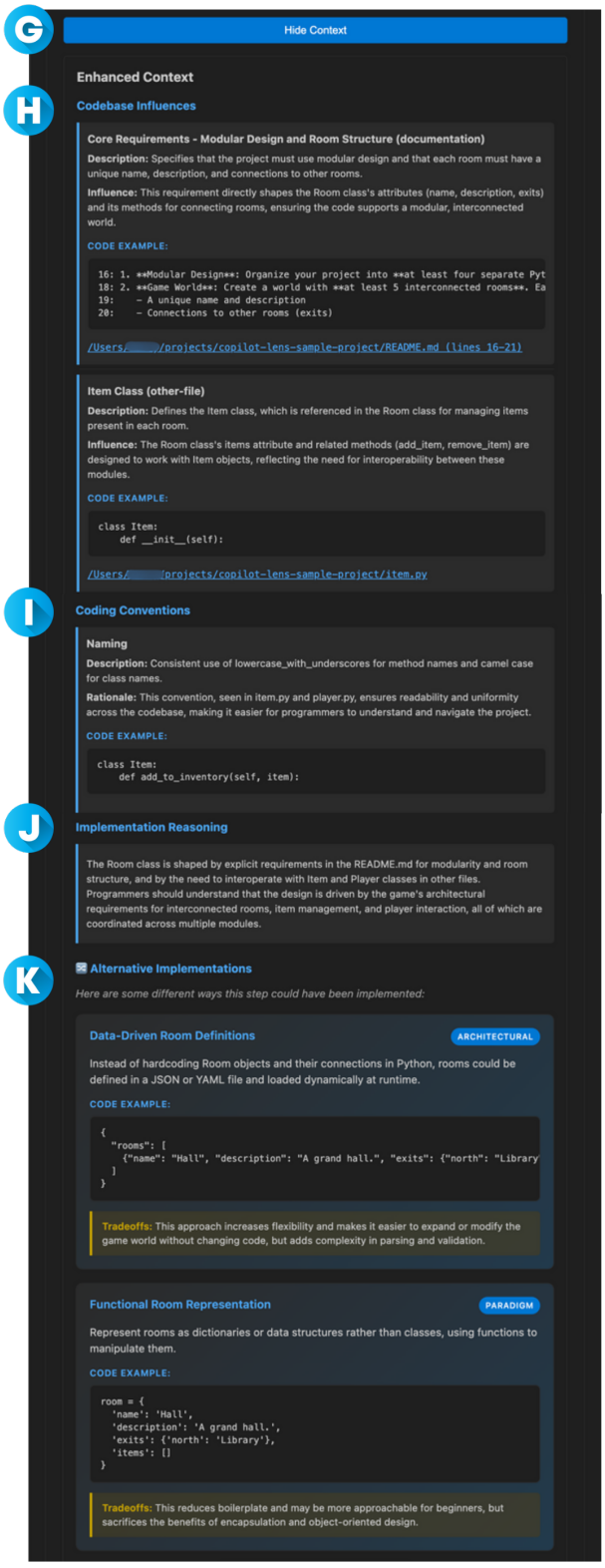


Fig. 3. Level 2: extended reasoning with influences, conventions, and alternatives.

VI. PRELIMINARY RESULTS AND FINDINGS

Our survey with 74 undergraduate participants produced several key insights into the use of AI coding assistants and the perceived need for explanation features. After cleaning invalid responses, 70 valid entries were analyzed.

A. Tool Familiarity and Usage

Participants reported high levels of familiarity with LLM-powered coding tools. The most common tools were ChatGPT (97%), Gemini (57%), Claude (43%), GitHub Copilot Inline (43%), and Cursor (32%). Less common tools included Copilot Agent (22%), Claude Code (11%), Gemini CLI (7%), and Windsurf IDE (3%). In terms of context, 60% reported using AI coding tools for coursework assignments, while 81% reported using them for debugging tasks.

B. Perceptions and Understanding Gap

While 94% of participants found AI tools to be time-saving and efficient, only half (50%) reported often leaving a coding session with a solid understanding of the resulting code. A large majority (84%) strongly agreed with the statement *"When I use an AI coding tool, I want to understand the reasoning behind the solutions it provides"* (mean = 4.27/5). This confirms a strong demand for design rationale explanations.

C. Pain Points

The most frequently reported pain points included:

- Overly complex or hard-to-debug code (66%).
- Use of obscure or deprecated libraries (63%).
- Poor adherence to coding conventions (38 mentions in open-ended responses).

These issues reflect a misalignment between students' mental models and the opaque outputs of coding agents.

D. Feature Helpfulness Ratings

When rating the helpfulness of explanation features (1 = not helpful, 5 = very helpful), the top results were:

- **Design Rationale:** 4.10 (highest rated).
- **Visual Code Diff:** 3.89.
- **Inline Confidence Highlighting:** 3.87.

E. Feature Prioritization Rankings

When asked to rank features by priority (1 = most important, 7 = least important), the top-ranked items were:

1) Dynamic UML Diagram (mean priority = 5.23).
2) Inline Confidence Highlighting (4.79).
3) Proactive Understanding Checker (4.13).

Interestingly, while *Change Summary* had a lower helpfulness score, it ranked highly in priority, suggesting that participants valued concise overviews even if they did not find them individually as transformative.

F. Implications

These findings validate the need for layered explanations in COPILOTLENS. Students want both quick, high-level summaries and deeper rationales that explain why code was written in a particular way. The results also highlight the importance of supporting comprehension alongside productivity, as most participants experienced gaps in understanding even when AI outputs were technically correct.

VII. CHALLENGES AND OBSTACLES

The development of COPILOTLENS presented several technical and methodological challenges. At the beginning of the project, we did not have a clear strategy for how to technically implement layered explanations on top of an AI coding assistant. Our first attempt was to build a lightweight, browser-based IDE environment and integrate our explanation overlay directly into it. While this allowed us to experiment with user interface concepts, it lacked the realism of a professional coding workflow and limited the complexity of tasks we could test.

Midway through the project, we identified the open-source agent *Kilo Code*, which provided multi-file editing capabilities in a more authentic development environment. By shifting our implementation to Kilo Code, we were able to anchor COPILOTLENS in a setting that reflected real-world coding tasks. This transition required significant re-engineering of our prototype, but it enabled us to design the two-level explanation views around actual agent-generated edits.

Another important challenge was balancing how much information COPILOTLENS should present. Too little detail risks reinforcing blind trust in AI, while too much overwhelms users. Our solution was the two-level design: a glanceable summary first, with deeper context only revealed on demand.

From a research perspective, ensuring high-quality survey data also posed difficulties. Some early responses had to be discarded as they contained no meaningful answers or were flagged as invalid due to unrealistically short completion times. To address this, we implemented de-identification and filtering scripts to clean the dataset before analysis.

Finally, there is an inherent challenge in generating explanations post-hoc. Since AI models do not expose their true internal reasoning, COPILOTLENS reconstructs plausible rationales from the output and project context. This raised questions of accuracy and interpretability, which we mitigated by grounding explanations in concrete evidence such as codebase references and explicit alternatives.

To support reproducibility, both stages of our implementation are available online. The prototype of the initial online IDE with integrated explanation overlay can be accessed at GitHub Repository (Online IDE Prototype). The COPILOTLENS prototype built on top of Kilo Code can be accessed at GitHub Repository (Kilo Code + COPILOTLENS).

VIII. NEXT STEPS AND FUTURE WORK

The following directions will guide the continuation of this project:

- Conduct semi-structured interviews with 66 survey volunteers to evaluate comprehension, error detection, and trust calibration in live coding tasks.
- Extend COPILOTLENS with adaptive explanation layers that adjust detail based on user expertise and task complexity.
- Integrate the prototype into widely used IDEs such as Visual Studio Code for broader accessibility.
- Run controlled experiments comparing COPILOTLENS against baseline coding agents, measuring comprehension scores, error detection, and cognitive load.
- Prepare a full-length conference or journal paper building on the COLM 2025 Workshop acceptance.

IX. RESOURCES AND TOOLS USED

The following resources and tools supported the development of COPILOTLENS:

- **Survey Platform:** Qualtrics for distribution and response collection.
- **Data Processing:** Python scripts (pandas, JSON handling) for de-identification, aggregation, and visualization.
- **Prototype Implementation:** Online IDE prototype (early stage) and Kilo Code agent (final stage).
- **Collaboration Tools:** Google Drive for document sharing, GitHub for version control, and Miro for research mapping.
- **Institutional Support:** King Abdullah University of Science and Technology (KAUST Academy) and the University of Toronto Data Sciences Institute (DSI).

ACKNOWLEDGMENTS

This project was supported by the King Abdullah University of Science and Technology (KAUST), KAUST Academy, the University of Toronto, the Data Sciences Institute (DSI), Core Labs, the National Transformation Data Program (NTDP), and the Alagil Foundation. We also thank the students who participated in our survey and provided valuable feedback for the prototype.

References

[1] R. Ye, Z. Zhang, B. Almazroua, and M. Liut, "Beyond autocomplete: Designing copilotlens towards transparent and explainable ai coding agents," in *Proceedings of the First Workshop on the Application of LLM Explainability to Reasoning and Planning (COLM 2025)*, 2025. [Online]. Available: https://arxiv.org/abs/2506.20062