Beyond Autocomplete: Designing CopilotLens Towards Transparent and Explainable AI Coding Agents

Runlong Ye
Computer Science, University of Toronto
Toronto, Ontario, Canada
harryye@cs.toronto.edu

Zeling Zhang
Computer Science, University of Toronto
Toronto, Ontario, Canada
zeling.zhang@mail.utoronto.ca

Boushra Almazroua*
King Abdullah University of Science and Technology
Thuwal, Jeddah, Saudi Arabia
boushra.almazroua@mail.utoronto.ca

Michael Liut
Mathematical and Computational Sciences, University of Toronto Mississauga
Mississauga, Ontario, Canada
michael.liut@utoronto.ca

Abstract

AI-powered code assistants are widely used to generate code completions, significantly boosting developer productivity. However, these tools typically present suggestions without explaining their rationale, leaving their decision-making process inscrutable. This opacity hinders developers' ability to critically evaluate outputs, form accurate mental models, and calibrate trust in the system. To address this, we introduce CopilotLens, a novel interactive framework that reframes code completion from a simple suggestion into a transparent, explainable interaction. CopilotLens operates as an explanation layer that reconstructs the AI agent's "thought process" through a dynamic, two-level interface. The tool aims to surface both high-level code changes and the specific codebase context influences. This paper presents the design and rationale of CopilotLens, offering a concrete framework and articulating expectations on deepening comprehension and calibrated trust, which we plan to evaluate in subsequent work.

1 Introduction

AI-powered assistants like GitHub Copilot[1] and more intelligent coding agents like Cursor[2] and WindSurf[3] are now essential tools in modern software development, evolving from simple code completion to autonomously executing complex, project-wide tasks Liang et al. (2024); Weisz et al. (2025). This trend is accelerating with the rise of asynchronous agents like Google's Jules[4], the rebranded OpenAI Codex[5], and Cursor's web-based agents[6], which abstract the programmer further away from the code. However, this utility comes at the cost of clarity Vasconcelos et al. (2025); Brachman et al. (2025). Mainstream assistants prioritize suggestion speed over reasoning, leaving their decision-making process inscrutable and

*Work done during research internship at the University of Toronto.
[1]https://github.com/features/copilot
[2]https://www.cursor.com/
[3]https://windsurf.com/
[4]https://jules.google/
[5]https://openai.com/index/introducing-codex/
[6]https://www.cursor.com/blog/agent-web/

forcing programmers to reverse-engineer the AI's intent. This opacity is a significant usability challenge that hinders the development of accurate mental models, suppresses critical evaluation of generated code Kazemitabaar et al. (2025); Mozannar et al. (2024), and fosters a fragile, uncalibrated trust Vaithilingam et al. (2022).

To address this critical gap between suggestion and comprehension, we argue for a paradigm shift toward explainable autocomplete Yan et al. (2024); Brachman et al. (2025) and introduce COPILOTLENS, an interactive explanation layer designed to reconstruct the "thought process" of an AI coding agent retrospectively. It reframes code completion from an isolated output into a transparent, reviewable event by surfacing the agent's plan, contextual evidence, and followed conventions.

This brief investment in transparency is designed to serve distinct developer needs. For *novice* student programmers, it transforms AI-assisted coding into a tangible learning experience, clearly exposing the underlying reasoning process to foster accurate mental models Prather et al. (2023). For *professional* developers, it aims to enable the rapid verification of complex multi-file suggestions, helping to prevent the accidental integration of subtle yet costly errors that prior large-scale studies have shown to be a persistent challenge Liang et al. (2024). More concrete use cases can be found in Appendix A.1.

The central design challenge we address is: ***How can the obscure reasoning process of an AI coding agent be redesigned into a transparent process that bridges the gap between an agent's internal logic and a developer's mental model, thereby supporting critical evaluation and fostering calibrated trust?***

We present COPILOTLENS as an initial design response to this question. We make the following contributions: (1) we articulate the challenges arising from the lack of transparency in current code assistants and formulate a set of design goals for creating more comprehensible tools; and (2) we introduce a dynamic, two-level explanation framework that exposes an agent's reasoning process, from high-level file modifications to the specific codebase influences behind a single change. By illuminating the agent's decision-making process, this work outlines a design framework and a research agenda for evaluating next-generation code assistants that not only write code, but also foster deeper understanding and more calibrated trust.

2 Related Work

Developers have long grappled with the cognitive challenges of understanding large and complex software systems. To manage this, they have historically relied on a variety of strategies to build and maintain a mental model of the code. Foundational techniques include comprehensive software documentation Von Mayrhauser & Vans (1995); LaToza et al. (2006), visual notations like UML to make system architecture tangible Koç et al. (2021), abstraction to hide implementation details Shaw et al. (1995), and program slicing to isolate relevant code Tip (1994). While effective, these traditional methods were designed to help humans comprehend *existing*, *human-written* codebases. The rise of AI-powered assistants has fundamentally altered this landscape, introducing a new kind of comprehension gap that these static methods are ill-equipped to address.

LLM-powered code assistants have rapidly transformed productivity gains for developers at every skill level Weber et al. (2024); Mozannar et al. (2024). Yet this productivity surge comes with a significant trade-off: opacity. Current LLM code assistants contribute code solutions without revealing the reasoning behind them Husein et al. (2025); Ferdowsi et al. (2024). This creates a new, active comprehension challenge: one not about understanding a static artifact, but about understanding the inscrutable *reasoning process* of an AI agent as it generates ever more complex programs. As a result, users often engage with these systems as "black boxes" and settle for a "good-enough" understanding Oldenburg & Søgaard (2025). This dynamic obstructs the formation of accurate mental models, discourages critical evaluation, and promotes uncalibrated trust Lee et al. (2025a); Wang et al. (2024).

Opacity introduces not only usability friction and trust concerns but also deeper cognitive burdens. Developers must often reverse-engineer the AI's intent post hoc, expending

cognitive effort to infer why certain completions were offered Weisz et al. (2021; 2025); Brown et al. (2024). This cognitive overhead is particularly challenging for novices, who may lack sufficient domain expertise to recognize subtle errors or inappropriate suggestions Kumar et al. (2024); Pammer-Schindler et al. (2025); Kazemitabaar et al. (2024). Over-reliance becomes common—and sometimes even conscious, among users Prather et al. (2023), as they can either blindly accept AI output or unknowingly propagate flawed code.

In response to these challenges, explainable AI (XAI) and Human-centered XAI (HXAI) have sought to enhance transparency and redefine explainability for LLMs Sun et al. (2022); Kim et al. (2025). Several promising paradigms have emerged. Token-level attribution allows users to trace model outputs to influential training examples, aiding provenance reasoning Lee et al. (2025b). Uncertainty visualization approaches offer complementary insights: by surfacing model confidence or expected edit likelihoods, they help users triage which suggestions warrant additional scrutiny Bhatt et al. (2021); Vasconcelos et al. (2025). However, these signals remain orthogonal to explaining the model's internal reasoning process, leaving the "why" behind generation opaque. Vasconcelos et al. (2023) emphasizes that different forms of explanation vary in their effectiveness, with explanations that make the AI's mistakes more salient being particularly effective at reducing overreliance. For example, self-consistency checks, where models verify their own outputs across multiple reasoning passes, have shown promise for detecting hallucinations or factual inconsistencies Leiser et al. (2024); Cheng et al. (2024a).

Despite such advances, current methods remain largely output-centric. They visualize or score the final artifact without exposing the reasoning that produced it, leaving users to reconstruct the logic from surface-level signals alone Miller (2019). Recent exploratory work has begun to suggest pathways for surfacing intermediate planning structures to bridge this divide Yen et al. (2024). COPILOTLENS, on the other hand, seeks to further close this gap by reframing code generation as an inspectable event. Rather than merely evaluating what the model produced, we seek to make visible how it arrived at that output, providing developers with actionable insight into the model's decision-making process and enabling richer, more trustworthy collaboration.

3 Design Goals

AI-powered coding assistants that prioritize speed of suggestion over clarity of reasoning create significant usability challenges that hinder effective human-AI collaboration Liang et al. (2024); Mozannar et al. (2024); Vasconcelos et al. (2025); Terragni et al. (2025). Mainstream AI tools, such as GitHub Copilot and Cursor, typically present a final code artifact with minimal insight into the agent's decision-making process (Figure 3, left). This opacity forces programmers to reverse-engineer the AI's intent Sergeyuk et al. (2025), which can lead to misaligned expectations, suppressed critical evaluation of generated code, and fragile, uncalibrated user trust Vaithilingam et al. (2022). Drawing from these established challenges, we formulate three design goals to guide the development of a more transparent and comprehensible code generation experience.

3.1 Challenge: Bridging the Gap Between AI Opacity and Developer Mental Models

A primary challenge with AI coding assistants is that developers lack accurate mental models of how the agent reaches its solutions Vaithilingam et al. (2022); Barke et al. (2023). Suggestions often feel "out-of-the-blue" because the agent's high-level plan, interpretation of user requests, and considered context remain hidden Ross et al. (2023). Consequently, developers interact with these tools as if they are inscrutable, overly confident partners.

The opacity of AI coding assistants actively reinforces flawed mental models and promotes ineffective developer behavior. This problem is worsened by minimalist interfaces that, while boosting productivity, hide the AI's reasoning by concealing which files it consulted or the steps it took. Forced to guess at the agent's capabilities, programmers are frequently inaccurate Vaithilingam et al. (2022). These misunderstandings can lead to poor prompts or

the blind acceptance of flawed code, stemming from an overestimation of the AI's contextual understanding and resulting in a fragile, uncalibrated trust Murillo et al. (2024).

***Design Goal G1**: Reveal the Agent's Thought Process.* A system should not just provide a final solution; instead, it should surface the breakdown of actions the agent took to arrive at that solution, making its process explicit and reviewable to bridge the user's mental model gap.

3.2 Challenge: Providing Deep Context to Scaffold Critical Evaluation

Simply revealing an agent's action sequence is insufficient for promoting deep comprehension Wood et al. (1976). Effective scaffolding must connect AI-generated code changes explicitly to the broader codebase context. Recent AI-driven approaches, such as the ephemeral UIs in computational notebooks, dynamically provide in-context explanations that enhance code comprehension Cheng et al. (2024b). Research shows that integrating architectural, API, and file-level context is critical for improving a user's ability to evaluate generated suggestions Wang et al. (2025).

Furthermore, effective scaffolds should provide layered, contextual rationales rather than isolated hints Hou et al. (2024), and must adaptively fade as developers gain expertise to avoid diminishing user autonomy Jennings & Muldner (2021). Hierarchical frameworks like CoLadder demonstrate that revealing intermediate planning steps improves comprehension by aligning the generated code with user intentions Yen et al. (2024). Similarly, tools like CodeCompass show that automatically surfacing relevant snippets from a repository aids developers in evaluating unfamiliar code Agrawal et al. (2024).

***Design Goal G2**: Support Informed Critical Evaluation.* The system should scaffold developers' critical evaluation by providing adaptive, deep contextual explanations linking specific code changes to project architecture, conventions, and design trade-offs.

3.3 Challenge: Making AI Reasoning Transparent to Foster Calibrated Trust

To trust AI-assisted development, developers need clear evidence that the system is not only correct but also makes logical decisions based on the project's context. Empirical findings by Brown et al. (2024) demonstrate that developers' trust in AI code suggestions significantly improves when they understand the assumptions and contextual reasoning behind the generated outputs, rather than relying solely on raw accuracy. Evaluations of contemporary AI-based assistants further show that generated methods frequently overlook project-specific constraints, highlighting the limitations of post-hoc verification as a sole mechanism for fostering trust Corso et al. (2024).

To address this gap, recent research has proposed "trust affordances", such as suggestion quality indicators and explicit usage analytics, enabling developers to form more accurate mental models of AI behavior by transparently communicating how suggestions align with established project norms and context Wang et al. (2024).

***Design Goal G3**: Make AI Reasoning Tangible and Verifiable.* The interface should explicitly present tangible evidence of the AI's reasoning, clearly linking each suggestion to the specific files, coding conventions, and architectural patterns used in its generation, thereby enabling developers to assess alignment with project standards immediately.

4 COPILOTLENS Probe: An Interactive and Modular Explanation Layer

To address our design goals, we designed and implemented COPILOTLENS (Figure 3, right). COPILOTLENS is an interactive explanation layer built on top of an existing open-source coding agent, Kilo Code[7]. At its core, COPILOTLENS intercepts the output of a coding agent *after* a task is complete to analyze and explain its actions. This design is guided by a specific rationale and is structured into a dynamic, two-level framework, which we detail below.

[7]https://kilocode.ai/

Identified Challenge in AI Code Generation	Manifestation in Current Tools (e.g., Copilot, Cursor)	COPILOTLENS Design Goal	Corresponding Feature(s)
Challenge 1: Opaque Reasoning Process & Misaligned Mental Models	Suggestions appear "out-of-the-blue" with no insight into the model's plan or decision points, leading to confusion Vaithilingam et al. (2022).	G1: Expose the Agent's Thought Process.	Level 1 Explanations: Streaming, sequential display of file modifications and interactive code highlighting.
Challenge 2: Lack of Contextual Understanding & Suppressed Critical Inquiry	Speed and fluency of suggestions discourage critical evaluation for subtle flaws or alternatives Mozannar et al. (2024).	G2: Support Informed Critical Evaluation.	Level 2 Explanations (On-Demand): Analysis of codebase influences, coding conventions, and alternative implementations.
Challenge 3: Making AI Reasoning Transparent to Foster Calibrated Trust	Users cannot directly assess suggestion reliability, as correctness is hidden, making trust fragile and uncalibrated Brown et al. (2024); Wang et al. (2024).	G3: Make Model Confidence and Correctness Tangible.	Level 2 Explanations (On-Demand): Explicitly linking code changes to project-specific artifacts (e.g., other files, documentation) to demonstrate process integrity.

Table 1: Design challenges, their manifestations in current AI code assistants, COPILOTLENS's three design goals, and the features that realize each goal.

4.1 Design Rationale: Post-Hoc, Model-Agnostic Explanation

The decision to perform post-hoc analysis is a pragmatic one. While direct, on-the-fly model interpretability is a primary goal of the broader XAI community, current techniques are not yet practical for our target workflow Zhao et al. (2023). State-of-the-art methods often require privileged "white-box" model access, which presents a difficult trade-off: it pushes developers to interact with open-weight models (i.e., LLama, Qwen, Mistral), while valuable for research, often underperform against leading proprietary APIs (i.e., GPT-5, Gemini 2.5, Claude 4) in complex coding tasks Jimenez et al. (2024). This would force an undesirable choice between developer productivity and model explainability. Beyond this, such techniques risk introducing significant latency or producing low-level, token-centric explanations that are difficult to map to a developer's high-level goals Burns et al. (2024). Such low-level signals often fail to provide meaningful insight into an agent's broader strategy, a long-standing challenge in XAI Miller (2019); Jain & Wallace (2019).

By instead reconstructing the agent's "thought process" from its final outputs, COPILOTLENS remains model-agnostic, allowing developers to use the most powerful coding agents available while still benefiting from a transparent, actionable explanation layer.

4.2 A Dynamic, Two-Level Explanation Framework

COPILOTLENS's central design principle is a two-level dynamic explanation framework designed to manage the inherent trade-off between informational quality and cognitive load, a key challenge in explainable AI (XAI) Miller (2019); Doshi-Velez & Kim (2017); Sweller (1988). It is designed as a research probe to investigate how programmers interact with AI-generated code when the "black box" is opened Sun et al. (2022). This two-level design is structured using the concepts of *explanatory* versus *exploratory* user interfaces (XUIs) Chromik & Butz (2021).

- **Level 1** functions as an *explanatory XUI*. It is designed for immediate, "at-a-glance" awareness, presenting a concise, post-hoc summary of the agent's actions. Its purpose is to quickly answer the question: "What just happened?".
- **Level 2** functions as an *exploratory XUI*. It is an on-demand, user-driven environment for deep investigation. Its purpose is to allow a developer to probe the rationale behind a specific change, answering the question: "Why was it done this way?".

By separating these two modes of interaction, the system supports both rapid sensemaking and deep reflective analysis, allowing developers to engage with the explanation at the level of detail appropriate for their current task.

4.3 Level 1: Post-Hoc Explanatory Summaries of Modifications (Figure 1)

In response to *G1 (Reveal the Agent's Process)*, COPILOTLENS presents a post-hoc explanatory summary of the agent's modifications. This initial level of explanation is designed for immediate situational awareness, automatically providing a per-file overview of the changes made. It features a side panel that sequentially displays each modified file with a concise summary of its purpose and significance, enabling rapid comprehension. Clicking on a summary navigates the user to the corresponding code changes (i.e., modified functions) in the code editor, creating a direct visual link between the explanation and the implementation.

4.4 Level 2: On-Demand Exploration of Development Rationale (Figure 2)

To address *G2 (Scaffold Deep Understanding)* and *G3 (Foster Calibrated Trust)*, the system provides a second, deeper layer of user-triggered exploration of the agent's rationale behind code modifications. Activated by the user, this level initiates an intensive, AI-powered analysis of a specific code change in relation to the entire codebase. The resulting insights are presented in distinct, evidence-based sections. The system first identifies ***Codebase Influences*** by surfacing the existing functional components that likely guided the agent's implementation, such as specific classes, functions, or documentation files. For each influence, it provides a description and a direct link to the source artifact as verifiable evidence.

This deeper analysis also articulates the ***Coding Conventions*** the agent adhered to, detecting language-specific architectural patterns, naming conventions, and stylistic choices, and other programming best practices and concepts. It provides a rationale for why a particular convention was applied and demonstrates its use with a concrete example of the generated code. To further encourage critical evaluation, the system can also propose ***Alternative Implementations***, describing different architectural or syntactic approaches and discussing their potential trade-offs. This form of contrastive explanation has been shown to improve independent decision-making Buçinca et al. (2025). By providing these specific, context-based insights, this exploratory level offers a tangible trust affordance, allowing developers to critically assess the AI's output against the project's established structure and practices.

5 Conclusion and Future Work

We presented COPILOTLENS, a framework that elevates AI assistants from opaque suggestion generators into transparent partners, by providing a two-level reasoning replay of *what* the agent did and *why*. This approach moves beyond simple explainability to support the developer's cognitive workflow, encouraging them to inspect, critique, and build calibrated trust.

Our future work will formally evaluate the framework against its core design goals for its intended users. For *students and novices*, we plan to explore how introducing productive "friction" and other learning science principles to enhance critical reflection Kazemitabaar et al. (2025), and embed learning opportunities directly within routine tool use Pammer-Schindler et al. (2025). For *professional developers*, we plan to focus on developers' mental model while mitigating the risk of cognitive overload from Level 2 analysis in large codebases by developing adaptive and configurable explanation interfaces. This will support rapid and collaborative verification within efficiency-driven workflows Weisz et al. (2025). This research will allow us to carefully investigate the trade-offs between explanation depth and cognitive load.

Acknowledgement

We acknowledge and thank the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number RGPIN-2024-04348]. The project is also supported by the Data Sciences Institute, University of Toronto.

References

Ekansh Agrawal, Omair Alam, Chetan Goenka, Medha Iyer, Isabela Moise, Ashish Pandian, and Bren Paul. Code compass: A study on the challenges of navigating unfamiliar codebases, 2024. URL https://arxiv.org/abs/2405.06271.

Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proc. ACM Program. Lang.*, 7(OOPSLA1), April 2023. doi: 10.1145/3586030. URL https://doi.org/10.1145/3586030.

Umang Bhatt, Javier Antorán, Yunfeng Zhang, Q. Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, Lama Nachman, Rumi Chunara, Madhulika Srikumar, Adrian Weller, and Alice Xiang. Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '21, pp. 401–413, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384735. doi: 10.1145/3461702.3462571. URL https://doi.org/10.1145/3461702.3462571.

Michelle Brachman, Arielle Goldberg, Andrew Anderson, Stephanie Houde, Michael Muller, and Justin D. Weisz. Towards personalized and contextualized code explanations. In *Adjunct Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization*, UMAP Adjunct '25, pp. 120–125, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713996. doi: 10.1145/3708319.3733681. URL https://doi.org/10.1145/3708319.3733681.

Adam Brown, Sarah D'Angelo, Ambar Murillo, Ciera Jaspan, and Collin Green. Identifying the factors that influence trust in ai code completion. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, AIware 2024, pp. 1–9, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706851. doi: 10.1145/3664646.3664757. URL https://doi.org/10.1145/3664646.3664757.

Zana Buçinca, Siddharth Swaroop, Amanda E. Paluch, Finale Doshi-Velez, and Krzysztof Z. Gajos. Contrastive explanations that anticipate human misconceptions can improve human decision-making skills. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713941. doi: 10.1145/3706598.3713229. URL https://doi.org/10.1145/3706598.3713229.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision, 2024. URL https://arxiv.org/abs/2212.03827.

Furui Cheng, Vilém Zouhar, Simran Arora, Mrinmaya Sachan, Hendrik Strobelt, and Mennatallah El-Assady. Relic: Investigating large language model responses using self-consistency. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3641904. URL https://doi.org/10.1145/3613904.3641904.

Ruijia Cheng, Titus Barik, Alan Leung, Fred Hohman, and Jeffrey Nichols. Biscuit: Scaffolding llm-generated code with ephemeral uis in computational notebooks. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 13–23, Sep. 2024b. doi: 10.1109/VL/HCC60511.2024.00012.

Michael Chromik and Andreas Butz. Human-xai interaction: A review and design principles for explanation user interfaces. In Carmelo Ardito, Rosa Lanzilotti, Alessio Malizia, Helen Petrie, Antonio Piccinno, Giuseppe Desolda, and Kori Inkpen (eds.), *Human-Computer Interaction – INTERACT 2021*, pp. 619–640, Cham, 2021. Springer International Publishing. ISBN 978-3-030-85616-8.

Vincenzo Corso, Leonardo Mariani, Daniela Micucci, and Oliviero Riganelli. Assessing ai-based code assistants in method generation tasks. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, pp. 380–381, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705021. doi: 10.1145/3639478.3643122. URL https://doi.org/10.1145/3639478.3643122.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017. URL https://arxiv.org/abs/1702.08608.

Kasra Ferdowsi, Ruanqianqian (Lisa) Huang, Michael B. James, Nadia Polikarpova, and Sorin Lerner. Validating ai-generated code with live programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642495. URL https://doi.org/10.1145/3613904.3642495.

Xinying Hou, Barbara J. Ericson, and Xu Wang. Integrating personalized parsons problems with multi-level textual explanations to scaffold code writing. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, pp. 1686–1687, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704246. doi: 10.1145/3626253.3635606. URL https://doi.org/10.1145/3626253.3635606.

Rasha Ahmad Husein, Hala Aburajouh, and Cagatay Catal. Large language models for code completion: A systematic literature review. *Computer Standards & Interfaces*, 92: 103917, 2025. ISSN 0920-5489. doi: https://doi.org/10.1016/j.csi.2024.103917. URL https://www.sciencedirect.com/science/article/pii/S0920548924000862.

Sarthak Jain and Byron C. Wallace. Attention is not explanation, 2019. URL https://arxiv.org/abs/1902.10186.

Jay Jennings and Kasia Muldner. When does scaffolding provide too much assistance? a code-tracing tutor investigation. *International Journal of Artificial Intelligence in Education*, 31(4):784–819, 2021. doi: 10.1007/s40593-020-00217-z. URL https://doi.org/10.1007/s40593-020-00217-z.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642773. URL https://doi.org/10.1145/3613904.3642773.

Majeed Kazemitabaar, Oliver Huang, Sangho Suh, Austin Z Henley, and Tovi Grossman. Exploring the design space of cognitive engagement techniques with ai-generated code for enhanced learning. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, IUI '25, pp. 695–714, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713064. doi: 10.1145/3708359.3712104. URL https://doi.org/10.1145/3708359.3712104.

Sunnie S. Y. Kim, Jennifer Wortman Vaughan, Q. Vera Liao, Tania Lombrozo, and Olga Russakovsky. Fostering appropriate reliance on large language models: The role of explanations, sources, and inconsistencies. In *Proceedings of the 2025 CHI Conference on*

Human Factors in Computing Systems, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713941. doi: 10.1145/3706598.3714020. URL https://doi.org/10.1145/3706598.3714020.

Hatice Koç, Ali Mert Erdoğan, Yousef Barjakly, and Serhat Peker. Uml diagrams in software engineering research: A systematic literature review. *Proceedings*, 74(1), 2021. ISSN 2504-3900. doi: 10.3390/proceedings2021074013. URL https://www.mdpi.com/2504-3900/74/1/13.

Harsh Kumar, Ilya Musabirov, Mohi Reza, Jiakai Shi, Xinyuan Wang, Joseph Jay Williams, Anastasia Kuzminykh, and Michael Liut. Guiding students in using llms in supported learning environments: Effects on interaction dynamics, learner performance, confidence, and trust. *Proc. ACM Hum.-Comput. Interact.*, 8(CSCW2), November 2024. doi: 10.1145/3687038. URL https://doi.org/10.1145/3687038.

Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pp. 492–501, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933751. doi: 10.1145/1134285.1134355. URL https://doi.org/10.1145/1134285.1134355.

Hao-Ping (Hank) Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. The impact of generative ai on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400713941. doi: 10.1145/3706598.3713778. URL https://doi.org/10.1145/3706598.3713778.

Seongmin Lee, Zijie J Wang, Aishwarya Chakravarthy, Alec Helbling, ShengYun Peng, Mansi Phute, Duen Horng Polo Chau, and Minsuk Kahng. Llm attributor: Interactive visual attribution for llm generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 29655–29657, 2025b.

Florian Leiser, Sven Eckhardt, Valentin Leuthe, Merlin Knaeble, Alexander Mädche, Gerhard Schwabe, and Ali Sunyaev. Hill: A hallucination identifier for large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642428. URL https://doi.org/10.1145/3613904.3642428.

Jenny T. Liang, Chenyang Yang, and Brad A. Myers. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3608128. URL https://doi.org/10.1145/3597503.3608128.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019. ISSN 0004-3702. doi: 10.1016/j.artint.2018.07.007. URL https://www.sciencedirect.com/science/article/pii/S0004370218305988.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. Reading between the lines: Modeling user behavior and costs in ai-assisted programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3641936. URL https://doi.org/10.1145/3613904.3641936.

Ambar Murillo, Alberto Elizondo, Sarah D'Angelo, Adam Brown, Ugam Kumar, Quinn Madison, and Andrew Macvean. Understanding and designing for trust in ai-powered developer tooling. *IEEE Software*, 41(6):23–28, Nov 2024. ISSN 1937-4194. doi: 10.1109/MS.2024.3439108.

Ninell Oldenburg and Anders Søgaard. Navigating the informativeness-compression trade-off in xai. *AI and Ethics*, 2025. doi: 10.1007/s43681-025-00733-5. URL https://doi.org/10.1007/s43681-025-00733-5.

Viktoria Pammer-Schindler, Michael Liut, and Tobias Ley. What if (everyday) technologies were designed for learning? towards "support for learning" as a design goal for every(day) technology. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI EA '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713958. doi: 10.1145/3706599.3719804. URL https://doi.org/10.1145/3706599.3719804.

James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers. *ACM Trans. Comput.-Hum. Interact.*, 31(1), November 2023. ISSN 1073-0516. doi: 10.1145/3617367. URL https://doi.org/10.1145/3617367.

Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, IUI '23, pp. 491–514, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701061. doi: 10.1145/3581641.3584037. URL https://doi.org/10.1145/3581641.3584037.

Agnia Sergeyuk, Ilya Zakharov, Ekaterina Koshchenko, and Maliheh Izadi. Human-ai experience in integrated development environments: A systematic literature review, 2025. URL https://arxiv.org/abs/2503.06195.

M. Shaw, R. DeLine, D.V. Klein, T.L. Ross, D.M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4):314–335, April 1995. ISSN 1939-3520. doi: 10.1109/32.385970.

Jiao Sun, Q. Vera Liao, Michael Muller, Mayank Agarwal, Stephanie Houde, Kartik Talamadupula, and Justin D. Weisz. Investigating explainability of generative ai for code through scenario-based design. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*, IUI '22, pp. 212–228, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391443. doi: 10.1145/3490099.3511119. URL https://doi.org/10.1145/3490099.3511119.

John Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285, 1988. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(88)90023-7. URL https://www.sciencedirect.com/science/article/pii/0364021388900237.

Valerio Terragni, Annie Vella, Partha Roop, and Kelly Blincoe. The future of ai-driven software engineering. *ACM Trans. Softw. Eng. Methodol.*, 34(5), May 2025. ISSN 1049-331X. doi: 10.1145/3715003. URL https://doi.org/10.1145/3715003.

Frank Tip. A survey of program slicing techniques. Technical report, NLD, 1994.

Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs experience: Evaluating the usability of code generation tools powered by large language models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391566. doi: 10.1145/3491101.3519665. URL https://doi.org/10.1145/3491101.3519665.

Helena Vasconcelos, Matthew Jörke, Madeleine Grunde-McLaughlin, Tobias Gerstenberg, Michael S. Bernstein, and Ranjay Krishna. Explanations can reduce overreliance on ai systems during decision-making. *Proc. ACM Hum.-Comput. Interact.*, 7(CSCW1), April 2023. doi: 10.1145/3579605. URL https://doi.org/10.1145/3579605.

Helena Vasconcelos, Gagan Bansal, Adam Fourney, Q. Vera Liao, and Jennifer Wortman Vaughan. Generation probabilities are not enough: Uncertainty highlighting in ai code completions. *ACM Trans. Comput.-Hum. Interact.*, 32(1), April 2025. ISSN 1073-0516. doi: 10.1145/3702320. URL https://doi.org/10.1145/3702320.

A. Von Mayrhauser and A.M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995. doi: 10.1109/2.402076.

Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. Investigating and designing for trust in ai-powered code generation tools. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '24, pp. 1475–1493, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704505. doi: 10.1145/3630106.3658984. URL https://doi.org/10.1145/3630106.3658984.

Yanlin Wang, Kefeng Duan, Dewu Zheng, Ensheng Shi, Fengji Zhang, Yanli Wang, Jiachi Chen, Xilin Liu, Yuchi Ma, Hongyu Zhang, et al. Towards an understanding of context utilization in code intelligence. *arXiv preprint arXiv:2504.08734*, 2025.

Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. Significant productivity gains through programming with large language models. *Proc. ACM Hum.-Comput. Interact.*, 8(EICS), June 2024. doi: 10.1145/3661145. URL https://doi.org/10.1145/3661145.

Justin D. Weisz, Michael Muller, Stephanie Houde, John Richards, Steven I. Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. Perfection not required? human-ai partnerships in code translation. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*, IUI '21, pp. 402–412, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380171. doi: 10.1145/3397481.3450656. URL https://doi.org/10.1145/3397481.3450656.

Justin D. Weisz, Shraddha Vijay Kumar, Michael Muller, Karen-Ellen Browne, Arielle Goldberg, Katrin Ellice Heintze, and Shagun Bajpai. Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI EA '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713958. doi: 10.1145/3706599.3706670. URL https://doi.org/10.1145/3706599.3706670.

D Wood, J S Bruner, and G Ross. The role of tutoring in problem solving. *J Child Psychol Psychiatry*, 17(2):89–100, Apr 1976. ISSN 0021-9630 (Print); 0021-9630 (Linking). doi: 10.1111/j.1469-7610.1976.tb00381.x.

Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. Ivie: Lightweight anchored explanations of just-generated code. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642239. URL https://doi.org/10.1145/3613904.3642239.

Ryan Yen, Jiawen Stefanie Zhu, Sangho Suh, Haijun Xia, and Jian Zhao. Coladder: Manipulating code generation via multi-level blocks. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706288. doi: 10.1145/3654777.3676357. URL https://doi.org/10.1145/3654777.3676357.

Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for large language models: A survey, 2023. URL https://arxiv.org/abs/2309.01029.

A Appendix

A.1 Use Case

A.1.1 Use Case for a Novice Student Programmer

A first-year student building a small web app receives an agent-generated set of code edits from COPILOTLENS. Before accepting, the student is shown a brief, post-hoc account of what changed and why, presented next to the relevant files; when uncertain, they open a deeper, on-demand rationale that ties the proposal to concrete evidence in the project (e.g., which existing modules appear to have guided the change) and contrasts it with plausible alternatives. This light-weight transparency nudges the student to verify intent, compare against course conventions, and accept only those edits they can justify, turning autocomplete from a "type-and-accept" action into a quick critique that reduces blind acceptance and fosters calibrated trust.

A.1.2 Use Case for a Professional Developer

A senior engineer reviewing an agent-authored refactor scans a concise summary that surfaces the key modifications and likely risk areas from COPILOTLENS, then selectively expands targeted explanations that connect each risky change to discoverable project context and articulate trade-offs versus reasonable alternatives. This evidence-backed pass functions as a rapid proofread: it helps confirm architectural fit, exposes subtle mismatches early, and avoids time wasted chasing dead ends, enabling the developer to merge what is sound, adjust what is misaligned, and reject what is unjustified, all without lowering the rigor of review.

A.2 COPILOTLENS Interface

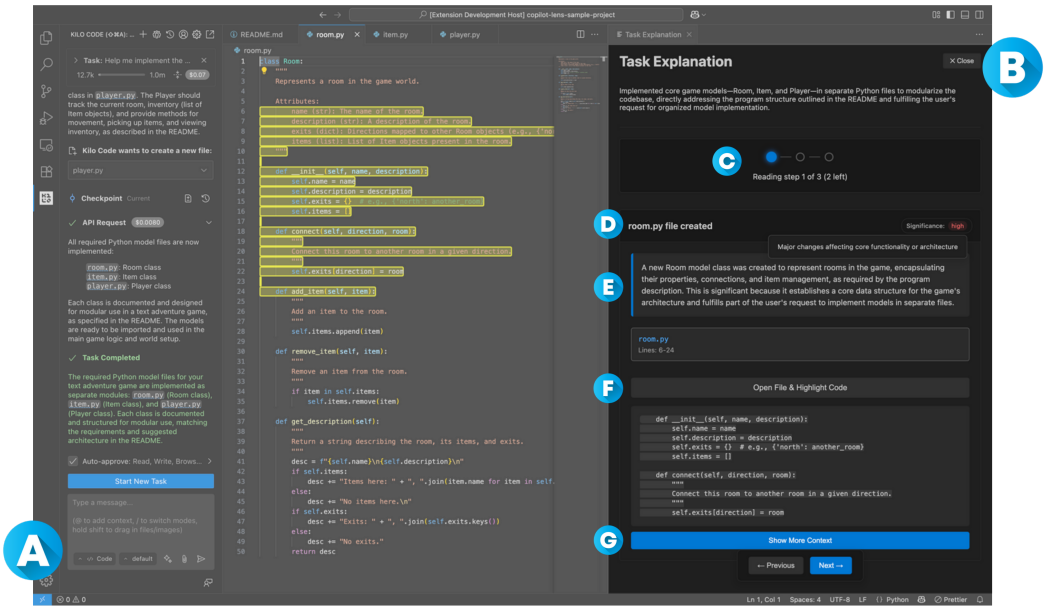

Figure 1: Main Interface for COPILOTLENS. (A) COPILOTLENS interface is presented after the AI coding agent has completed its assigned task. (B) The main explanation view for COPILOTLENS, which provides a post-hoc, two-level analysis of the agent's actions. The default Level 1 explanation offers a summary of what was changed, including (C) a visual navigator to step through each file modification, (D) the title of the current modification and its significance, (E) an AI-generated summary of the change's purpose and significance, and (F) a preview of the implemented code and interactive highlights. (G) The user can click to trigger the on-demand Level 2 analysis, which provides an extended explanation.

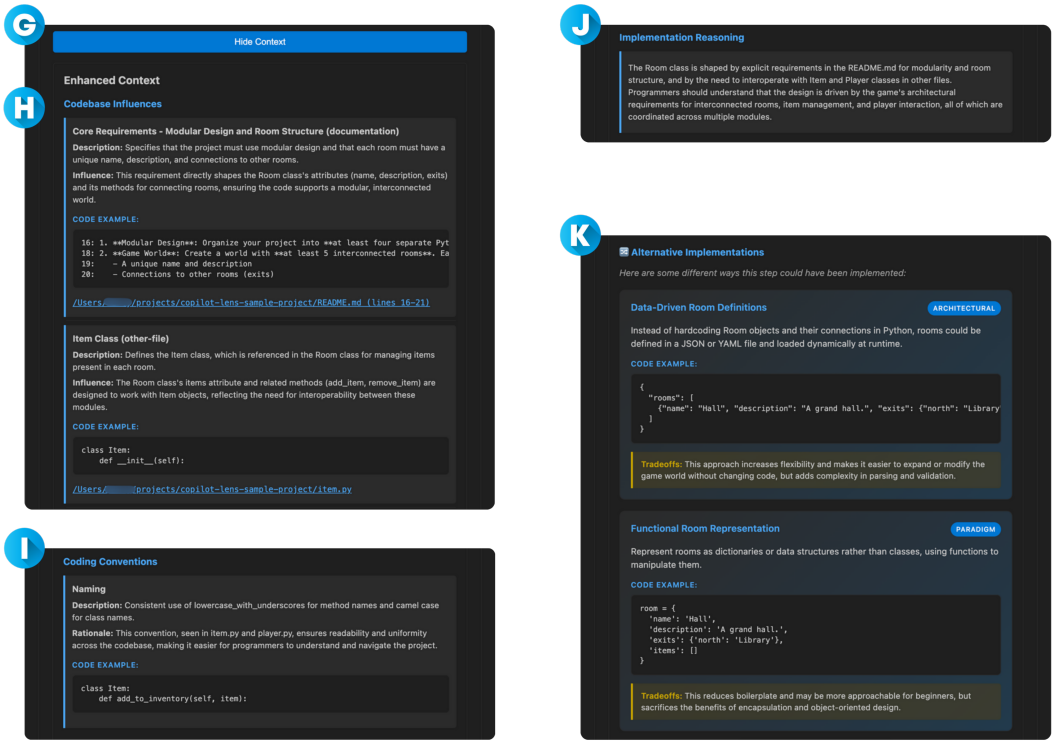

Figure 2: Extended Explanation Interface for CopilotLens. (G) This Level 2 explanation view is shown after the user clicks the button to see more context. It provides a deeper analysis of the agent's work, presenting (H) a list of "Codebase Influences" that shows which existing project files or documentation were influencing the generated code, with file linking and highlighting upon user clicking the hyperlink, (I) a section on "Coding Conventions" that explains style and pattern choices, (J) a section with detailed "Implementation Reasoning" explaining the rationale behind the changes, and (K) a list of "Alternative Implementations" describing other ways the task could have been accomplished, along with their respective tradeoffs.

A.3 CopilotLens Comparison with Popular AI Coding Agents

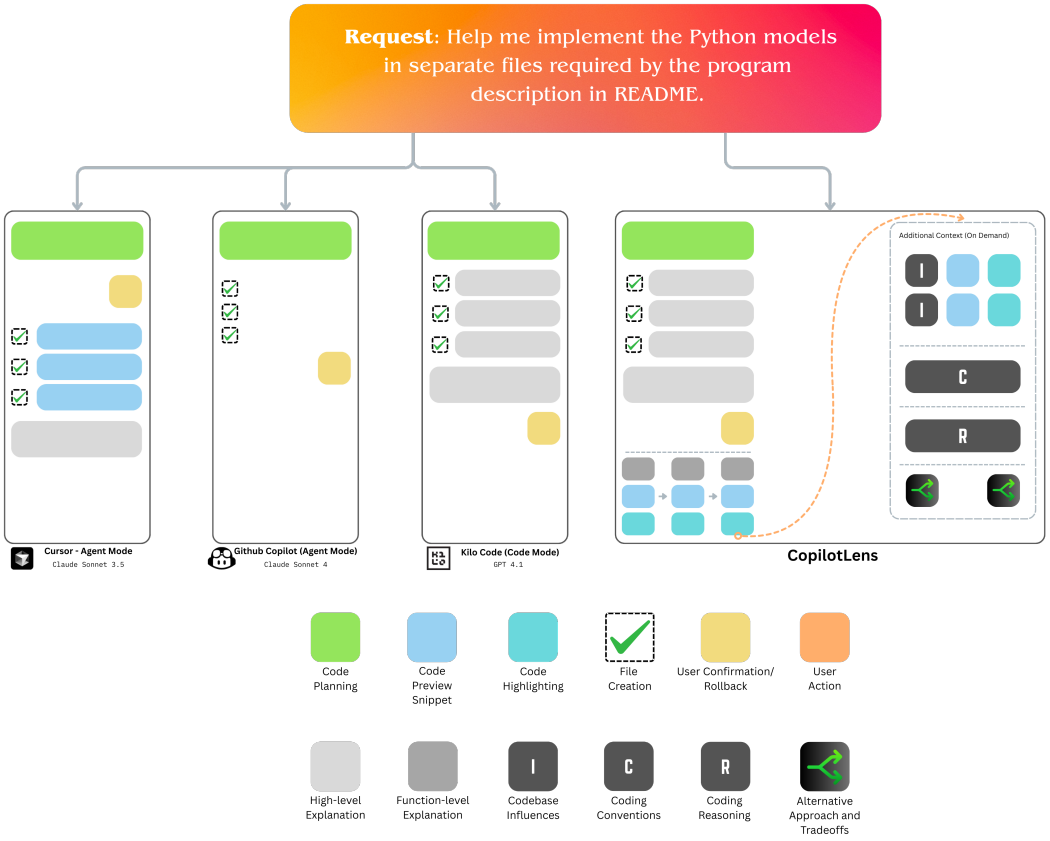

Figure 3: A Comparison of AI Coding Agent Interaction Panels. Mainstream tools like Cursor, GitHub Copilot, and open-source coding agent Kilo Code (left) present a final code artifact with minimal process visibility and explanations. In contrast, COPILOTLENS (right) reconstruct the agent's "thought process" through a two-level explanation, surfacing important codebase influence (I), coding convention (C), coding reasoning (R), and alternative approach to consider, to support critical evaluation and foster calibrated trust.