Novelty Assessment Report

Paper: EditBench: Evaluating LLM Abilities to Perform Real-World Instructed Code Edits
PDF URL: https://openreview.net/pdf?id=FtL9eEmU6v
Venue: ICLR 2026 Conference Submission
Year: 2026
Report Generated: 2025-12-27

Abstract

Instructed code editing, where LLMs directly modify a developer's existing code based on a user instruction, is becoming a widely used interaction mode in AI coding assistants. However, few benchmarks directly evaluate this capability and current datasets often rely on artificial sources. We introduce EditBench, a benchmark for evaluating LLM code editing capabilities grounded in real-world usage, i.e.,~user instructions and code contexts collected in the wild. EditBench comprises of 545 problems, multiple natural and programming languages, and a diverse set of real-world use cases, ranging from resolving errors to adding features. EditBench introduces context-dependent problems that require the model to understand code context, highlighted code, and cursor position in addition to the user instruction. We evaluate 40 diverse LLMs and observe that EditBench is a challenging set of problems where only 3 models score over 60\%. We find that model performance varies across different categories of user instructions. Further, we find that varying levels of contextual information greatly affect task success rate, with performance varying up to 11\%, indicating the importance of evaluating with realistic context.

Core Task Landscape

This paper addresses: **Instructed Code Editing with Large Language Models**
A total of **50 papers** were analyzed and organized into a taxonomy with **30 categories**.

Taxonomy Overview

The research landscape has been organized into the following main categories:

- **Code Editing Frameworks and Benchmarks**
- **Prompting and Instruction Strategies**
- **Instruction Tuning for Code Editing**
- **Iterative Refinement and Feedback Mechanisms**
- **Code Generation and Optimization**
- **Domain-Specific Code Editing**
- **Security and Vulnerability Repair**
- **Developer-LLM Interaction Studies**
- **Code Editing Architectures**
- **Multimodal and Cross-Domain Editing**
- ... and 2 more categories

Complete Taxonomy Tree

- Instructed Code Editing with Large Language Models Survey Taxonomy
- Code Editing Frameworks and Benchmarks
  - Real-World Code Editing Evaluation ★ (3 papers)
  - [0] EditBench: Evaluating LLM Abilities to Perform Real-World Instructed Code Edits (Anon et al., 2026) View paper
  - [35] CodeEditorBench: Evaluating code editing capability of LLMs (J Guo, 2025) View paper
  - [47] Codeeditorbench: Evaluating code editing capability of large language models (Guo Jiawei, 2024) View paper
  - Synthetic Dataset Generation for Code Editing (2 papers)
  - [16] Unprecedented code change automation: The fusion of llms and transformation by example (Bryksin, 2024) View paper
  - [42] Generating High-Quality Datasets for Code Editing via Open-Source Language Models (Zhang Zekai, 2025) View paper
- Prompting and Instruction Strategies
  - Prompt Engineering for Code Modification (3 papers)
  - [4] Exploring Direct Instruction and Summary-Mediated Prompting in LLM-Assisted Code Modification (Tang, 2025) View paper
  - [8] Prompting llms for code editing: Struggles and remedies (Nam, 2025) View paper
  - [30] Enhancing computer programming education with llms: A study on effective prompt engineering for python code generation (Wang Tianyu, 2024) View paper
  - Requirements Clarification and Intent Alignment (2 papers)
  - [31] ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification (Fangwen Mu, 2024) View paper
  - [33] NeuroSync: Intent-Aware Code-Based Problem Solving via Direct LLM Understanding Modification (Zhang Wen-shuo, 2025) View paper
  - Multi-Stage and Planning-Based Generation (2 papers)
  - [14] Self-Planning Code Generation with Large Language Models (Xue Jiang, 2024) View paper
  - [27] Multi-stage guided code generation for Large Language Models (Yewei Han, 2025) View paper

- Instruction Tuning for Code Editing
  - Specialized Code Editing Instruction Tuning (3 papers)
  - [15] NextCoder: Robust Adaptation of Code LMs to Diverse Code Edits (T Aggarwal, 2025) View paper
  - [28] Bridging the Editing Gap in LLMs: FineEdit for Precise and Targeted Text Modifications (Yiming Zeng, 2025) View paper
  - [37] Instructcoder: Instruction tuning large language models for code editing (Chen Hui, 2024) View paper
  - Versatile Code Task Instruction Tuning (3 papers)
  - [13] InverseCoder: Unleashing the Power of Instruction-Tuned Code LLMs with Inverse-Instruct (Yutong Wu, 2024) View paper
  - [43] WaveCoder: Widespread And Versatile Enhancement For Code Large Language Models By Instruction Tuning (Hu Wen-Xiang, 2023) View paper
  - [45] CursorCore: Assist Programming through Aligning Anything (Jiang Hao, 2024) View paper
  - Knowledge Editing in Language Models (2 papers)
  - [36] InstructEd: Soft-Instruction Tuning for Model Editing with Hops (Han Xiaoqi, 2024) View paper
  - [39] InstructEdit: Instruction-based Knowledge Editing for Large Language Models (Zhang, 2024) View paper
- Iterative Refinement and Feedback Mechanisms
  - Execution-Based Iterative Repair (3 papers)
  - [11] The Art of Repair: Optimizing Iterative Program Repair with Instruction-Tuned Models (Hort, 2025) View paper
  - [20] Perfcodegen: Improving performance of llm generated code with execution feedback (Yun Peng, 2025) View paper
  - [21] Rectifier: Code translation with corrector via llms (Xin Yin, 2024) View paper
  - Multi-Session Collaborative Coding (1 papers)
  - [22] From Tools to Teammates: Evaluating LLMs in Multi-Session Coding Interactions (Campos, 2025) View paper
  - Speculative and Accelerated Editing (1 papers)
  - [49] Reuse or Generate? Accelerating Code Editing via Edit-Oriented Speculative Decoding (P Wang, 2025) View paper
- Code Generation and Optimization
  - Repository-Level and Multi-Agent Code Generation (2 papers)
  - [5] Codeplan: Repository-level coding using llms and planning (Ramakrishna Bairi, 2024) View paper
  - [41] Self-Organized Agents: A LLM Multi-Agent Framework toward Ultra Large-Scale Code Generation and Optimization (Ishibashi Yoichi, 2024) View paper
  - Optimization Code Generation (1 papers)
  - [32] LLaMoCo: Instruction Tuning of Large Language Models for Optimization Code Generation (Ma Zeyuan, 2024) View paper
  - Code Translation and Transpilation (2 papers)
  - [3] Verified code transpilation with LLMs (Sahil Bhatia, 2024) View paper
  - [34] Multi-IaC-Eval: Benchmarking Cloud Infrastructure as Code Across Multiple Formats (Davidson, 2025) View paper
  - Decompilation and Reverse Engineering (1 papers)
  - [48] Refining Decompiled C Code with Large Language Models (Wong Wai-Kin, 2023) View paper
- Domain-Specific Code Editing
  - Hardware Description Language Generation (1 papers)
  - [10] HaVen: Hallucination-Mitigated LLM for Verilog Code Generation Aligned with HDL Engineers (Yiyao Yang, 2025) View paper
  - Safety-Critical Code Generation (1 papers)
  - [9] On Simulation-Guided LLM-based Code Generation for Safe Autonomous Driving Software (Nouri Ali, 2025) View paper
- Security and Vulnerability Repair (2 papers)
  - [24] Security Degradation in Iterative AI Code Generation--A Systematic Analysis of the Paradox (Himanshu, 2025) View paper
  - [46] LLM-Powered Code Vulnerability Repair with Reinforcement Learning and Semantic Reward (Islam, 2024) View paper
- Developer-LLM Interaction Studies (1 papers)
  - [7] Developer-llm conversations: An empirical study of interactions and generated code quality (Zou Ying, 2025) View paper
- Code Editing Architectures
  - Executable Code Action Frameworks (1 papers)
  - [6] Executable Code Actions Elicit Better LLM Agents (Wang, 2024) View paper
  - Tool Instruction Optimization (1 papers)
  - [25] EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction (Chen, 2024) View paper
  - Code World Models (1 papers)
  - [23] Generating Code World Models with Large Language Models Guided by Monte Carlo Tree Search (Nicola Dainese, 2024) View paper
  - Knowledge-Oriented Programming (1 papers)
  - [38] Diakop: Dialogue-based knowledge-oriented programming for neural-symbolic knowledge base question answering (Zhicheng Lee, 2024) View paper
- Multimodal and Cross-Domain Editing
  - Visual and Creative Content Editing (3 papers)
  - [1] ChatGarment: Garment Estimation, Generation and Editing via Large Language Models (Siyuan Bian, 2024) View paper
  - [2] Spellburst: A node-based interface for exploratory creative coding with natural language prompts (Han Jenny, 2023) View paper
  - [12] AutoVFX: Physically Realistic Video Editing from Natural Language Instructions (Hao-Yu Hsu, 2024) View paper
  - Structured Data Editing (2 papers)
  - [18] PlotEdit: Natural Language-Driven Accessible Chart Editing in PDFs via Multimodal LLM Agents (Kanika Goswami, 2025) View paper
  - [19] WikiTableEdit: A Benchmark for Table Editing by Natural Language Instruction (Li Zheng, 2024) View paper
  - Multimodal Synthetic Data Generation (1 papers)
  - [17] Scaling Text-Rich Image Understanding via Code-Guided Synthetic Multimodal Data Generation (Callison-Burch, 2025) View paper
  - Speech Understanding and Editing (1 papers)
  - [29] Ming-UniAudio: Speech LLM for Joint Understanding, Generation and Editing with Unified Representation (Canxiang Yan, 2025) View paper
  - Time Series Editing (1 papers)
  - [50] Instruction-based Time Series Editing (Qiu Jiaxing, 2025) View paper

- Educational Applications (2 papers)
  - [26] Generative AI in introductory programming instruction: Examining the assistance dilemma with LLM-based code generators (Eric Poitras, 2024) View paper
  - [44] Natural language outlines for code: Literate programming in the llm era (Kensen Shi, 2025) View paper
- Code Quality and Correctness Analysis (1 papers)
  - [40] Code Generation Using LLMs (Tenneti Lekhya Sri Durga, 2025) View paper

Narrative

Core task: instructed code editing with large language models. The field has organized itself around several complementary dimensions. At the highest level, researchers have developed Code Editing Frameworks and Benchmarks to measure real-world performance, alongside Prompting and Instruction Strategies that explore how to elicit precise edits from LLMs. Instruction Tuning for Code Editing focuses on adapting models through supervised fine-tuning, while Iterative Refinement and Feedback Mechanisms investigate multi-turn correction loops. Parallel branches address Code Generation and Optimization (often overlapping with editing when models rewrite for efficiency), Domain-Specific Code Editing (targeting specialized languages or contexts), and Security and Vulnerability Repair. Additional threads examine Developer-LLM Interaction Studies to understand human workflows, Code Editing Architectures that propose novel model designs, Multimodal and Cross-Domain Editing for tasks beyond pure text, Educational Applications, and Code Quality and Correctness Analysis to verify outputs.

Within this landscape, a particularly active line of work centers on building realistic benchmarks that capture the complexity of professional code modification. EditBench[0] exemplifies this direction by evaluating LLMs on authentic editing scenarios drawn from real repositories, emphasizing localized changes rather than generation from scratch. This approach contrasts with earlier efforts like CodeEditorBench[35] and CodeEditorBench[47], which also target practical editing but may differ in dataset construction or task granularity. Meanwhile, works such as Executable Code Actions[6] and CursorCore[45] explore how to integrate editing into interactive development environments, and studies like Developer LLM Conversations[7] examine the back-and-forth between programmers and assistants. EditBench[0] sits squarely in the real-world evaluation cluster, sharing its emphasis on ecological validity with these neighboring benchmarks while contributing a distinct set of tasks and metrics that highlight the nuances of instruction-following during localized code modifications.

Related Works in Same Category

The following **2 sibling papers** share the same taxonomy leaf node with the original paper:

1. CodeEditorBench: Evaluating code editing capability of LLMs

Authors: J Guo, Z Li, X Liu, K Ma, T Zheng, et al. (6 authors total) | **Year/Venue**: 2025 | **URL**: View paper

Abstract

â¦ To comprehensively assess LLMâs code editing performance across the four scenarios, we construct OJ based on the hustoj (Haobin, 2012). The system processes LLM-generated â¦

Relationship Analysis

Both papers belong to the Real-World Code Editing Evaluation category, focusing on benchmarking LLM code editing capabilities using authentic developer contexts. EditBench and CodeEditorBench overlap in evaluating instructed code edits with real-world scenarios, including bug fixing and feature modifications, and both use test harnesses to assess correctness. However, EditBench distinguishes itself by collecting data from actual in-the-wild developer interactions via a VS Code extension (545 problems from 458 users), capturing contextual elements like highlighted code and cursor position, while CodeEditorBench constructs 7,961 problems from existing competitive programming sources (LeetCode, CodeNet, etc.) with synthetically inserted errors and focuses on four specific editing scenarios (debug, translate, polish, requirement switch).

---

2. Codeeditorbench: Evaluating code editing capability of large language models

Authors: Guo Jiawei, Jiawei Guo, Li ZiMing, Ziming Li, Liu Xue-ling, et al. (35 authors total) | **Year/Venue**: 2024 | **URL**: View paper

Abstract

Large Language Models (LLMs) for code are rapidly evolving, with code editing emerging as a critical capability. We introduce CodeEditorBench, an evaluation framework designed to rigorously assess the performance of LLMs in code editing tasks, including debugging, translating, polishing, and requirement switching. Unlike existing benchmarks focusing solely on code generation, CodeEditorBench emphasizes real-world scenarios and practical aspects of software development. We curate diverse coding c...

Relationship Analysis

Both papers belong to the Real-World Code Editing Evaluation category, focusing on benchmarking LLM code editing capabilities using authentic developer contexts. EditBench and CodeEditorBench overlap in evaluating instructed code edits with real-world scenarios, diverse programming languages, and comprehensive test harnesses. However, EditBench uniquely collects data from in-the-wild developer interactions via a VS Code extension with contextual features (highlighted code, cursor position), while CodeEditorBench constructs synthetic editing tasks from competitive programming sources across four predefined categories (debug, translate, polish, requirement switch) without capturing actual developer workflows.

Contributions Analysis

Overall novelty summary. The paper introduces EditBench, a benchmark for evaluating LLM code editing capabilities using real-world user instructions and code contexts. It resides in the 'Real-World Code Editing Evaluation' leaf, which contains only three papers including this one. This represents a relatively sparse research direction within the broader taxonomy of 50 papers across 30 topics, suggesting that authentic, production-grounded code editing benchmarks remain an emerging area despite the maturity of adjacent fields like general code generation and synthetic dataset creation.

The taxonomy reveals that EditBench's immediate neighbors include 'Synthetic Dataset Generation for Code Editing' (two papers) and broader categories like 'Prompting and Instruction Strategies' (five papers across three sub-leaves) and 'Instruction Tuning for Code Editing' (eight papers across three sub-leaves). While the field has invested heavily in training methods and prompt engineering, the scarcity of real-world evaluation frameworks indicates a gap between model development and ecologically valid assessment. EditBench bridges this gap by emphasizing authentic developer interactions rather than commit-based or artificially constructed tasks.

Among the three contributions analyzed, the core EditBench benchmark examined ten candidates with zero refutations, suggesting novelty in its specific design. The VS Code extension for data collection examined ten candidates and found one refutable prior work, indicating some overlap with existing data collection tools. The context-dependent evaluation mechanism examined only two candidates with no refutations, though the limited search scope (22 total candidates across all contributions) means this analysis captures a snapshot rather than exhaustive coverage of the literature.

Based on the top-22 semantic matches examined, EditBench appears to occupy a relatively novel position within a sparse research direction. The benchmark's emphasis on real-world instructions, contextual information (highlighted code, cursor position), and diverse

use cases distinguishes it from sibling papers in the same leaf, though the limited search scope means additional related work may exist beyond the candidates analyzed here.

This paper presents **3 main contributions**, each analyzed against relevant prior work:

Contribution 1: EditBench benchmark for real-world instructed code editing

Description: The authors present EditBench, a benchmark comprising 545 problems sourced from real-world developer interactions. It evaluates LLMs on instructed code editing tasks using authentic user instructions, code contexts, highlighted code segments, and cursor positions across multiple natural and programming languages.

This contribution was assessed against **10 related papers** from the literature. Papers with potential prior art are analyzed in detail with textual evidence; others receive brief assessments.

---

1. SVGEditBench V2: A Benchmark for Instruction-based SVG Editing

URL: View paper

Brief Assessment

SVGEditBench[58] focuses on instruction-based SVG editing for vector graphics, not general code editing. The domains, data sources (emoji datasets vs. developer interactions), and evaluation contexts are fundamentally different.

---

2. Codeeditorbench: Evaluating code editing capability of large language models

URL: View paper

Brief Assessment

CodeEditorBench[47] focuses on evaluating code editing across four specific scenarios (debug, translate, polish, requirement switch) using synthetically generated problems from competitive programming sources, while the original paper's EditBench evaluates real-world instructed code edits collected in-the-wild from actual developers using a VS Code extension with contextual information like highlighted code and cursor position.

---

3. Feature requests-based recommendation of software refactorings

URL: View paper

Brief Assessment

Feature Request Refactoring[57] focuses on recommending software refactorings based on feature requests from issue trackers, not on evaluating LLM code editing capabilities with real-world user instructions and code contexts.

---

4. Opportunities and challenges in repeated revisions to pull-requests: An empirical study

URL: View paper

Brief Assessment

Pull Request Revisions[55] focuses on collaborative code review practices in pull requests, not on benchmarking LLM capabilities for instructed code editing tasks with user instructions and code contexts.

---

5. Automated recommendation of software refactorings based on feature requests

URL: View paper

Brief Assessment

Automated Refactoring Recommendation[56] focuses on recommending software refactorings based on feature requests using machine learning, not on evaluating LLM code editing capabilities with real-world user instructions and code contexts as EditBench does.

---

6. Editeval: An instruction-based benchmark for text improvements

URL: View paper

Brief Assessment

EditEval[52] focuses on text editing tasks (paraphrasing, neutralization, simplification, fluency) for natural language content, not code editing. The candidate addresses iterative text improvements in domains like Wikipedia and news articles, while the original paper evaluates LLM capabilities for instructed code editing with real-world developer interactions.

---

7. Large Language Models of Code Fail at Completing Code with Potential Bugs

URL: View paper

Brief Assessment

Code Fail Bugs[54] focuses on code completion with potential bugs in the context, not instructed code editing with user instructions. The candidate addresses a different problem (completing buggy code) rather than editing code based on natural language instructions from real-world developer interactions.

---

8. VectorEdits: A Dataset and Benchmark for Instruction-Based Editing of Vector Graphics

URL: View paper

Brief Assessment

VectorEdits[59] focuses on instruction-based editing of vector graphics (SVG images), not code editing. The domains, tasks, and evaluation contexts are fundamentally different.

---

9. A study of update request comments in Stack Overflow answer posts

URL: View paper

Brief Assessment

Update Request Comments[53] focuses on Stack Overflow comment analysis to identify update requests in answer posts, not on benchmarking LLM code editing capabilities with real-world user instructions and code contexts.

---

10. Refactorbench: Evaluating stateful reasoning in language agents through code

URL: View paper

Brief Assessment

RefactorBench[51] focuses on multi-file code refactoring tasks requiring compositional reasoning across dependencies, while EditBench evaluates single-file instructed code edits based on user instructions with highlighted code segments. The benchmarks address different interaction modalities and task complexities in code editing.

Contribution 2: VS Code extension for in-the-wild data collection

Description: The authors built a VS Code extension that mimics existing code editing tools to gather live, in-the-wild data from nearly 500 users. This extension collects user-written instructions, associated code context, and user votes between model responses during real coding workflows.

This contribution was assessed against **10 related papers** from the literature. Papers with potential prior art are analyzed in detail with textual evidence; others receive brief assessments.

---

1. CognitIDE: An IDE Plugin for Mapping Physiological Measurements to Source Code

URL: View paper

Brief Assessment

CognitIDE[68] is an IntelliJ plugin for collecting physiological measurements (eye gaze, body sensors) mapped to source code, not a VS Code extension for collecting code editing data and user instructions from developers.

---

2. How far are AI-powered programming assistants from meeting developers' needs?

URL: View paper

Brief Assessment

AI Programming Assistants[70] focuses on evaluating existing AI coding assistants through user studies with screen recording and code evaluation, not on building an extension that mimics code editing tools for collecting user instructions and model responses during real coding workflows.

---

3. In-ide code generation from natural language: Promise and challenges

URL: View paper

Prior Art Analysis

In IDE Generation[64] demonstrates prior work on building an IDE plugin for collecting in-the-wild code generation data from developers. The candidate paper explicitly describes developing 'a plugin for the IDE that implements a hybrid of code generation and code retrieval functionality, and orchestrate virtual environments to enable collection of many user events.' This directly parallels the original paper's contribution of building a VS Code extension to collect user-written instructions, code context, and user responses during real coding workflows. Both papers collect data from developers performing actual programming tasks, gather user interactions with model responses, and release the collected data for research purposes.

Evidence

Evidence 1 - **Rationale**: Both papers describe building IDE extensions/plugins to collect in-the-wild data from developers during actual coding tasks. The original paper collects 'user-written instructions, associated code context, and user votes' while the candidate collects 'many user events' from developers completing programming tasks. - **Original**: we source our problems by developing a vs code extension that mimics existing instructed code editing tools from github copilot and cursor. as developers use the extension, we gather a live, in-the-wild dataset containing user-written instructions, associated code context, and user votes between pai... - **Candidate**: we first develop a plugin for the ide that implements a hybrid of code generation and code retrieval functionality, and orchestrate virtual environments to enable collection of many user events. we ask developers with various backgrounds to complete 14 python programming tasks ranging from basic fil...

---

4. KOALA: a Configurable Tool for Collecting IDE Data When Solving Programming Tasks

URL: View paper

Brief Assessment

KOALA[65] is designed for JetBrains IDEs (IntelliJ IDEA, PyCharm, CLion), not VS Code. The original paper specifically built a VS Code extension, while KOALA[65] targets a completely different IDE ecosystem and does not challenge the novelty of creating a VS Code extension for code editing data collection.

---

5. Mind the Metrics: Patterns for Telemetry-Aware In-IDE AI Application Development using the Model Context Protocol (MCP)

URL: View paper

Brief Assessment

Mind the Metrics[63] focuses on telemetry-aware IDE development using MCP for AI application observability and prompt optimization, not on collecting code editing data from developers for benchmark creation.

---

6. Enhancing Incremental Dataflow Analysis in an IDE

URL: View paper

Brief Assessment

Incremental Dataflow Analysis[69] focuses on optimizing dataflow analysis within IDEs through graph-based representations (SP-Graph), not on building extensions for collecting user coding data or evaluating LLM code editing capabilities.

---

7. Developer Behaviors in Validating and Repairing LLM-Generated Code Using IDE and Eye Tracking

URL: View paper

Brief Assessment

Developer Behaviors Validation[67] focuses on studying developer behaviors when validating/repairing LLM-generated code through IDE and eye-tracking in controlled lab settings, not on building extensions for collecting real-world code editing data from users in their natural workflows.

---

8. A Study on Developer Behaviors for Validating and Repairing LLM-Generated Code Using Eye Tracking and IDE Actions

URL: View paper

Brief Assessment

Developer Behaviors Validation[66] focuses on studying developer behaviors when validating and repairing LLM-generated code through eye tracking and IDE actions, not on building data collection extensions for code editing workflows.

---

9. AntiCopyPaster: An Open-Source Ecosystem for Just-in-time Code Duplicates Extraction

URL: View paper

Brief Assessment

AntiCopyPaster[71] is an IntelliJ IDEA plugin for detecting code duplicates and suggesting refactorings, not a VS Code extension for collecting user instruction data during code editing workflows.

---

10. CodeWatcher: IDE Telemetry Data Extraction Tool for Understanding Coding Interactions with LLMs

URL: View paper

Brief Assessment

CodeWatcher[62] focuses on logging fine-grained interaction events (insertions, deletions, copy-paste) for behavioral analysis of code generation tool usage, whereas the original paper's extension specifically collects instructed code editing data (user instructions, code context, highlighted segments, and user votes between model responses) for benchmark construction.

Contribution 3: Context-dependent evaluation with highlighted code and cursor position

Description: The benchmark uniquely incorporates multiple contextual elements beyond the user instruction, including the full code file, highlighted code regions, and cursor position. This makes EditBench the first benchmark to evaluate instructed code edits with this combination of features.

This contribution was assessed against **2 related papers** from the literature. Papers with potential prior art are analyzed in detail with textual evidence; others receive brief assessments.

---

1. Eliph: Effective visualization of code history for peer assessment in programming education

URL: View paper

Brief Assessment

Eliph[61] focuses on visualizing code history for peer assessment in programming education, not on evaluating LLM code editing capabilities. The highlighted code and cursor position in Eliph[61] serve a different purpose—enabling students to track code changes over time during peer review—rather than providing context for instructed code edits as in the original paper.

---

2. Eyes on code: A study on developers' code navigation strategies

URL: View paper

Brief Assessment

Eyes on Code[60] focuses on developers' code navigation strategies using eye-tracking during bug-fixing tasks, not on evaluating LLM code editing capabilities with highlighted code and cursor position as contextual inputs for instructed edits.

Appendix: Text Similarity Detection

No high-similarity text segments were detected across any compared papers.

References

- [0] EditBench: Evaluating LLM Abilities to Perform Real-World Instructed Code Edits View paper
- [1] ChatGarment: Garment Estimation, Generation and Editing via Large Language Models View paper
- [2] Spellburst: A node-based interface for exploratory creative coding with natural language prompts View paper
- [3] Verified code transpilation with LLMs View paper
- [4] Exploring Direct Instruction and Summary-Mediated Prompting in LLM-Assisted Code Modification View paper
- [5] Codeplan: Repository-level coding using llms and planning View paper
- [6] Executable Code Actions Elicit Better LLM Agents View paper
- [7] Developer-llm conversations: An empirical study of interactions and generated code quality View paper
- [8] Prompting llms for code editing: Struggles and remedies View paper
- [9] On Simulation-Guided LLM-based Code Generation for Safe Autonomous Driving Software View paper
- [10] HaVen: Hallucination-Mitigated LLM for Verilog Code Generation Aligned with HDL Engineers View paper
- [11] The Art of Repair: Optimizing Iterative Program Repair with Instruction-Tuned Models View paper
- [12] AutoVFX: Physically Realistic Video Editing from Natural Language Instructions View paper
- [13] InverseCoder: Unleashing the Power of Instruction-Tuned Code LLMs with Inverse-Instruct View paper
- [14] Self-Planning Code Generation with Large Language Models View paper
- [15] NextCoder: Robust Adaptation of Code LMs to Diverse Code Edits View paper
- [16] Unprecedented code change automation: The fusion of llms and transformation by example View paper
- [17] Scaling Text-Rich Image Understanding via Code-Guided Synthetic Multimodal Data Generation View paper
- [18] PlotEdit: Natural Language-Driven Accessible Chart Editing in PDFs via Multimodal LLM Agents View paper
- [19] WikiTableEdit: A Benchmark for Table Editing by Natural Language Instruction View paper
- [20] Perfcodegen: Improving performance of llm generated code with execution feedback View paper
- [21] Rectifier: Code translation with corrector via llms View paper
- [22] From Tools to Teammates: Evaluating LLMs in Multi-Session Coding Interactions View paper
- [23] Generating Code World Models with Large Language Models Guided by Monte Carlo Tree Search View paper
- [24] Security Degradation in Iterative AI Code Generation--A Systematic Analysis of the Paradox View paper
- [25] EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction View paper
- [26] Generative AI in introductory programming instruction: Examining the assistance dilemma with LLM-based code generators View paper
- [27] Multi-stage guided code generation for Large Language Models View paper
- [28] Bridging the Editing Gap in LLMs: FineEdit for Precise and Targeted Text Modifications View paper
- [29] Ming-UniAudio: Speech LLM for Joint Understanding, Generation and Editing with Unified Representation View paper
- [30] Enhancing computer programming education with llms: A study on effective prompt engineering for python code generation View paper
- [31] ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification View paper
- [32] LLaMoCo: Instruction Tuning of Large Language Models for Optimization Code Generation View paper
- [33] NeuroSync: Intent-Aware Code-Based Problem Solving via Direct LLM Understanding Modification View paper
- [34] Multi-IaC-Eval: Benchmarking Cloud Infrastructure as Code Across Multiple Formats View paper
- [35] CodeEditorBench: Evaluating code editing capability of LLMs View paper
- [36] InstructEd: Soft-Instruction Tuning for Model Editing with Hops View paper
- [37] Instructcoder: Instruction tuning large language models for code editing View paper

- [38] Diakop: Dialogue-based knowledge-oriented programming for neural-symbolic knowledge base question answering View paper
- [39] InstructEdit: Instruction-based Knowledge Editing for Large Language Models View paper
- [40] Code Generation Using LLMs View paper
- [41] Self-Organized Agents: A LLM Multi-Agent Framework toward Ultra Large-Scale Code Generation and Optimization View paper
- [42] Generating High-Quality Datasets for Code Editing via Open-Source Language Models View paper
- [43] WaveCoder: Widespread And Versatile Enhancement For Code Large Language Models By Instruction Tuning View paper
- [44] Natural language outlines for code: Literate programming in the llm era View paper
- [45] CursorCore: Assist Programming through Aligning Anything View paper
- [46] LLM-Powered Code Vulnerability Repair with Reinforcement Learning and Semantic Reward View paper
- [47] Codeeditorbench: Evaluating code editing capability of large language models View paper
- [48] Refining Decompiled C Code with Large Language Models View paper
- [49] Reuse or Generate? Accelerating Code Editing via Edit-Oriented Speculative Decoding View paper
- [50] Instruction-based Time Series Editing View paper
- [51] Refactorbench: Evaluating stateful reasoning in language agents through code View paper
- [52] Editeval: An instruction-based benchmark for text improvements View paper
- [53] A study of update request comments in Stack Overflow answer posts View paper
- [54] Large Language Models of Code Fail at Completing Code with Potential Bugs View paper
- [55] Opportunities and challenges in repeated revisions to pull-requests: An empirical study View paper
- [56] Automated recommendation of software refactorings based on feature requests View paper
- [57] Feature requests-based recommendation of software refactorings View paper
- [58] SVGEditBench V2: A Benchmark for Instruction-based SVG Editing View paper
- [59] VectorEdits: A Dataset and Benchmark for Instruction-Based Editing of Vector Graphics View paper
- [60] Eyes on code: A study on developers' code navigation strategies View paper
- [61] Eliph: Effective visualization of code history for peer assessment in programming education View paper
- [62] CodeWatcher: IDE Telemetry Data Extraction Tool for Understanding Coding Interactions with LLMs View paper
- [63] Mind the Metrics: Patterns for Telemetry-Aware In-IDE AI Application Development using the Model Context Protocol (MCP) View paper
- [64] In-ide code generation from natural language: Promise and challenges View paper
- [65] KOALA: a Configurable Tool for Collecting IDE Data When Solving Programming Tasks View paper
- [66] A Study on Developer Behaviors for Validating and Repairing LLM-Generated Code Using Eye Tracking and IDE Actions View paper
- [67] Developer Behaviors in Validating and Repairing LLM-Generated Code Using IDE and Eye Tracking View paper
- [68] CognitIDE: An IDE Plugin for Mapping Physiological Measurements to Source Code View paper
- [69] Enhancing Incremental Dataflow Analysis in an IDE View paper
- [70] How far are AI-powered programming assistants from meeting developers' needs? View paper
- [71] AntiCopyPaster: An Open-Source Ecosystem for Just-in-time Code Duplicates Extraction View paper