RECODE-H: A BENCHMARK FOR RESEARCH CODE DEVELOPMENT WITH INTERACTIVE HUMAN FEEDBACK

Chunyu Miao[1], **Henry Peng Zou**[1,*,†], **Yangning Li**[2,*,†], **Yankai Chen**[1,3,4,†], **Yibo Wang**[1,†], **Fangxin Wang**[1], **Yifan Li**[5], **Wooseong Yang**[1], **Bowei He**[4,6], **Xinni Zhang**[5], **Dianzhi Yu**[5], **Hanchen Yang**[1], **Hoang H Nguyen**[1], **Yue Zhou**[1], **Jie Yang**[1], **Jizhou Guo**[7], **Wenzhe Fan**[1], **Chin-Yuan Yeh**[8], **Panpan Meng**[9], **Liancheng Fang**[1], **Jinhu Qi**[1], **Wei-Chieh Huang**[1], **Zhengyao Gu**[1], **Yuwei Han**[1], **Langzhou He**[1], **Yuyao Yang**[1], **Yinghui Li**[2], **Hai-Tao Zheng**[2], **Xue Liu**[3,4], **Irwin King**[5], **Philip S. Yu**[1]

[1]University of Illinois Chicago [2]Tsinghua University [3]McGill University [4]MBZUAI
[5]The Chinese University of Hong Kong [6]City University of Hong Kong
[7]Shanghai Jiao Tong University [8]National Taiwan University [9]Xi'an Jiaotong University

{cmiao8, pzou3, ychen588, ywang633}@uic.edu

ABSTRACT

Large language models (LLMs) show promise for supporting scientific research implementation, yet their ability to generate correct, executable code remains limited. Existing works largely adopt one-shot settings, ignoring the iterative and feedback-driven nature of realistic scientific research workflows. To address this gap, we present RECODE, a benchmark of 102 tasks from research papers and repositories that evaluates LLMs through multi-turn interactions with human feedback. It includes structured instructions, unit tests, and a five-level feedback hierarchy to reflect realistic researcher–agent collaboration. We further present ReCodeAgent, a framework that integrates feedback into an iterative code-generation process. Experiments[1] with leading LLMs, including GPT-5, Claude-Sonnet-4, DeepSeek-V3.1, and Gemini 2.5, show substantial performance gains with richer feedback, while also highlighting ongoing challenges in generating complex research code. RECODE-H establishes a foundation for developing adaptive, feedback-driven LLM agents in scientific research implementation.

1 INTRODUCTION

Large language models (LLMs) have been increasingly adopted across the scientific research pipeline, assisting tasks from ideation to writing (Zhang et al., 2025b; Si et al., 2024; Chen et al., 2025b; Li et al., 2025b; Wu et al., 2025b; Zou et al., 2025b; Zhang et al., 2025a). However, generating correct and executable research code remains a difficult problem, not only because it requires long-range reasoning and robust verification (Padigela et al., 2025; Starace et al., 2025; Zhu et al., 2025c), but also because the input contexts in research settings are often complex, indirect, and noisy. Research papers describe methods through high-level narratives, mathematical formulas, and domain-specific conventions, while leaving many implementation details implicit. As a result, translating these fragmented and underspecified descriptions into functional code remains a fundamental challenge for current LLMs (Li et al., 2025c;a).

Existing benchmarks for research code generation (Zheng et al., 2023; Sun et al., 2023; Toledo et al., 2025; Hua et al., 2025) primarily evaluate models in a non-interactive setting, where they are expected to produce correct code in a single response. This design neglects the crucial role of human feedback in realistic workflows: on the one hand, users often cannot fully specify their requirements in one shot. On the other hand, LLMs rarely generate perfectly aligned code on the first attempt (Zou

[1]We will make our dataset and code publicly available.

Domain	Benchmark	Repo	Unittest	Feedback	Task
General Code Generation	BigCodeBench	✗	✓	✗	Function Code Generation
	ConvCodeWorld	✗	✓	Verbal Feedback	Function Code Generation
	MINT	✗	✓	Lazy User Feedback	Function Code Generation
	InterCode	✗	✓	Execution Feedback	Function Code Generation
Research Code Generation	MLE-bench	✗	✗	✗	Machine Learning Engineering
	MLAgentBench	✓	✓	✗	Machine Learning Engineering
	PaperBench	✓	✗	✗	Reproduce ICML Papers
	SciReplicate-Bench	✓	✓	✗	Code generation for research
	RECODE-H	✓	✓	Hierarchical Researcher Feedback	Code generation for research

Table 1: Overview of benchmarks and their characteristics. Categories are abbreviated for compactness: Our benchmark introduces a structured feedback hierarchy (Section 3.2) to support systematic evaluation of interactive research code generation under progressively richer forms of guidance.

et al., 2025b). In practice, effective implementation relies on multi-turn interactions, where models must accurately interpret and leverage iterative user feedback (Li et al., 2024d). However, current benchmarks focus solely on end-to-end correctness, without evaluating models' capabilities in interactive refinement.

To fill this gap, we introduce **RECODE-H** (**Research CO**de **DE**velopment), a benchmark designed to evaluate how LLMs generate and refine research code through interactive feedback with human researchers. The benchmark consists of 102 tasks drawn from real research papers across machine learning, natural language processing, computer vision, and computational science, each paired with its original codebase. Unlike simple function-completion tasks, these tasks focus on repository-level code generation, requiring models to implement classes, functions, or modules that correspond to methodological descriptions in real research codebases. Building RECODE-H is challenging (Cao et al., 2025), as it requires aligning paper descriptions with code implementations, selecting representative tasks, and maintaining expert-level quality at scale. We address these challenges through a hybrid LLM-assisted and human-curated pipeline. RECODE-H has three key features:

- **PhD-level difficulty and expert annotation.** All tasks are drawn from real research projects and manually curated to ensure clarity, realism, and high annotation quality.
- **Feedback-level controlled difficulty.** Task difficulty is systematically controlled through the structured feedback hierarchy as introduced in Section 3.2, enabling fine-grained evaluation of models' ability to leverage multi-turn feedback guidance.
- **Research method focus.** Tasks center on the faithful implementation of research methods, typically requiring the development of several functions up to entire classes, rather than isolated function completion.

In our experiments, we evaluate seven leading LLMs, including GPT-5, DeepSeek-V3.1, Claude-Sonnet-4, and the Gemini family, on RECODE-H. We also introduce ReCode Agent, a stronger baseline that leverages human feedback through structured multi-turn interactions. ReCode Agent progressively integrates diagnostic signals, refinement instructions, and correction feedback to guide model revisions, providing a more faithful simulation of real research workflows. Experimental results show that all models substantially benefit from interactive feedback, with even minimal diagnostic signals nearly doubling pass rates and recall compared to the no-feedback setting. For example, GPT-5's recall improves from 29.4% without feedback to 71.6% with the most detailed feedback, while DeepSeek-V3.1 shows a similar jump from 10.8% to 70.6%. Larger models, such as GPT-5 and DeepSeek-V3.1, demonstrate stronger adaptation to progressively richer feedback, while Claude-Sonnet-4 and Gemini lag behind. Increasing feedback granularity not only improves success rates but also accelerates convergence, enabling stronger models to solve tasks in fewer turns. Error analysis further reveals that failures are dominated by misinterpretation of paper instructions and gaps in domain knowledge, rather than syntax or integration issues. These findings highlight the effectiveness of ReCode Agent and structured feedback, establishing a solid baseline for future work on research code generation.

Overall, our work makes the following contributions:

- We introduce **RECODE-H**, the first benchmark for *multi-turn interactive research code generation*, providing a high-quality dataset that evaluates how LLMs generate and refine code under iterative human feedback.

- We propose **ReCode Agent**, a strong baseline that effectively leverages structured human feedback through iterative interaction to implement and refine research code in realistic repository settings.
- Through extensive experiments on seven leading LLMs, we reveal key limitations of current models and demonstrate the effectiveness of our method, offering insights for building future feedback-driven research agents.

2 RELATED WORK

LLMs for Research Development. LLM-based agents have been applied across the scientific workflow, from survey writing and idea generation to paper review (Wang et al., 2024b; Si et al., 2025; Weng et al., 2024; He et al., 2026; Zhang et al., 2025a). Early benchmarks focused on end-to-end workflows (Toledo et al., 2025; Kon et al., 2025; Chan et al., 2024; Chen et al., 2024c; Starace et al., 2025; Edwards et al., 2025), but implementation remains a major bottleneck: current models struggle to translate textual descriptions into executable code. Recent datasets such as SciReplicate (Xiang et al., 2025), ResearchCodeBench (Hua et al., 2025), and LMR-Bench (Yan et al., 2025) shift toward code-level evaluation. Our work builds on this direction by introducing a qualitatively new evaluation paradigm by incorporating **multi-turn, feedback-driven** interactions. RECODE-H is the first benchmark to systematically assess an agent's ability to iteratively revise and refine research code through a structured hierarchy of feedback, reflecting the iterative cycles of diagnosis, correction, and alignment that characterize real scientific practice.

Interactive Code Generation. In software engineering (SWE), benchmarks span a wide spectrum, from function-level code generation tasks (Chen et al., 2021; Austin et al., 2021; Chen et al., 2023a; Zhuo et al., 2024; Hendrycks et al., 2021; Athiwaratkun et al., 2022) to repository-level generation (Jimenez et al., 2023; Li et al., 2024a;b; Ding et al., 2023). Most evaluations adopt a one-shot setting, where models produce solutions without further interaction with humans. More recently, as summarized in Table 1, Han et al. (2025); Wang et al. (2023b); Yang et al. (2023) have explored multi-turn interactive code generation by introducing various forms of feedback. However, these tasks remain limited to relatively simple function-level generation within the SWE domain. In contrast, our benchmark targets research code generation at the repository level, where the nature and complexity of feedback differ substantially from those in existing SWE-oriented benchmarks.

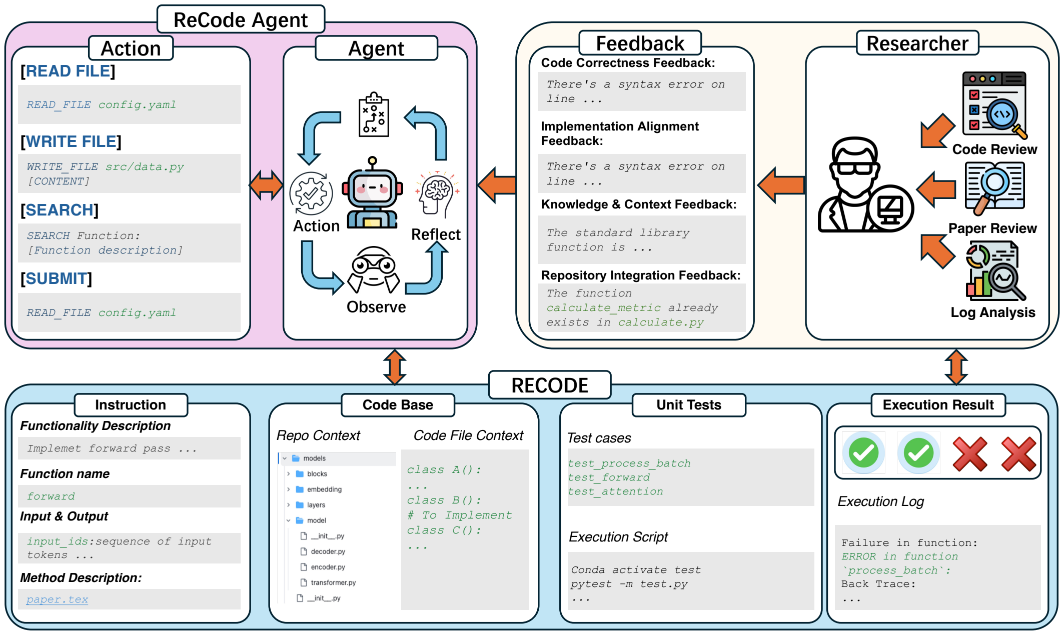
3 BENCHMARK

RECODE is designed to evaluate the ability of large language models to generate and refine research code in realistic, feedback-driven workflows. It consists of 102 tasks drawn from research papers and their corresponding repositories, each paired with structured instructions, explanatory comments, and unit tests to ensure reproducibility and reliability. The detailed benchmark statistics are provided in Appendix B.

3.1 BENCHMARK CONSTRUCTION

We developed a collaborative framework for constructing RECODE-H. To support this process, we assembled an annotation team of 26 annotators. All annotators are Ph.D.-level researchers with at least one publication at top-tier computer science conferences and are familiar with the target methods or algorithms, as well as their implementation code. The annotation process follows a multi-step pipeline: (1) paper and code selection, (2) annotation of explanatory comments for code, (3) construction of code generation instructions, and (4) development of unit tests. This design ensures both the reliability and reproducibility of the benchmark. An annotated example could be found in Appendix N.

Paper and Code Selection. All papers and repositories in RECODE-H were manually curated by expert annotators across diverse AI research domains, including machine learning algorithms, NLP, computer vision, graph learning, time-series modeling, diffusion models, and recommendation systems (detailed distribution in Appendix B). The paper selection process follows a strict curation protocol to ensure realism, reproducibility, and alignment between research descriptions and implementation details.



```
READ_FILE config.yaml
```

```
WRITE_FILE src/data.py
[CONTENT]
```

```
SEARCH Function:
[Function description]
```

```
READ_FILE config.yaml
```

```
There's a syntax error on
line ...
```

```
There's a syntax error on
line ...
```

```
The standard library
function is ...
```

```
The function
calculate_metric already
exists in calculate.py
```

```
Implemet forward pass ...
```

```
forward
```

```
input_ids:sequence of input
tokens ...
```

```
paper.tex
```

```
class A():
...
class B():
# To Implement
class C():
...
```

```
test_process_batch
test_forward
test_attention
```

```
Conda activate test
pytest -m test.py
...
```

✓ ✓ ✗ ✗

```
Failure in function:
ERROR in function
`process_batch`:
Back Trace:
...
```

Figure 1: Illustration of the RECODE-H workflow, where LLM agents iteratively generate, test, and refine research code through structured researcher feedback.

Specifically, we include only papers that satisfy the following criteria: (1) the paper was published in 2023 or later; (2) the venue is a top-tier computer science conference (CVPR, ICML, NeurIPS, or ICLR)(3) the authors provide an official open-source repository that is correct and executable; and (4) the paper and repository exhibit a clear correspondence between methodological descriptions and code structure. Annotators verify each repository by running the official scripts to ensure the targeted components execute successfully within standard hardware constraints (less than 24 GB GPU memory). Only repositories meeting all criteria are included in RECODE-H, ensuring that the benchmark reflects realistic, well-specified, and executable research code tasks. The full paper list is revealed in Appendix K.

Annotation of Explanatory Comments for Code. The correspondence between code and paper descriptions is often distributed across multiple functions or classes, and the mapping is not always explicit. To address this, we add explanatory comments that clarify the relationship between the code and its associated paper. These comments will help produce consistent feedback, which is crucial for the reproducible evaluation process of RECODE-H.

To reduce annotation workload and ensure consistent comment content, we adopt a human-LLM collaborative approach. We employ `Gemini-2.5-Pro` to generate explanatory comments, taking as input both the paper text and the identified code segments. The comments primarily focus on three aspects: (1) the correspondence between the code and the paper description, (2) any discrepancies between the paper and the actual implementation, and (3) implementation details present in the code but absent from the paper. To ensure reliability, all generated comments are subsequently reviewed and validated for correctness.

Construction of Code Generation Instructions. Once the correspondence between the code and the paper description is established, we construct detailed code generation instructions. Each instruction specifies the target function name, its intended functionality, and the input and output parameters, including their names, data types, and semantic roles. For functionality descriptions, we mainly focus on illustrating the relationship between the function and the paper description and add necessary explanations to the functionality, ensuring the instructions remain faithful to the original research context.

To promote consistency, we employ `Gemini-2.5-Pro` to generate an initial draft of the instruction in a standardized format. These drafts are then manually reviewed and refined by annotators to guarantee accuracy, clarity, and alignment with the source paper and code.

Development of Unit Tests. Unit tests are the core mechanism for evaluating functional correctness in RECODE-H. While LLMs (`Gemini-2.5-Pro`) were used only to draft initial candidate test cases; all tests were subsequently reviewed, corrected, and finalized by expert annotators. Annotators verified every test for correctness and completeness and enforced a strict 80% code coverage requirement for the canonical implementation. This ensures that the tests exercise the majority of logic branches rather than relying on superficial or syntactic checks. To accommodate complex research code functions, including those that internally perform complex computations, such as training updates, optimization routines, or data preprocessing workflows, we adopt a general differential testing strategy. Rather than evaluating full end-to-end system behavior, each unit test executes both the canonical implementation and the model-generated implementation under identical, fixed-seed conditions and compares their intermediate or final outputs. This comparison ensures functional equivalence within appropriate numerical tolerances, allowing us to verify the correctness of individual computational steps even when the overall task involves inherently dynamic or stateful processes. This approach provides a reliable and scalable mechanism for validating algorithmic correctness across a broad range of complex research tasks. These differential tests provide a reliable and reproducible evaluation framework across all task types in RECODE-H.

3.2 Feedback Hierarchy

In real-world scenarios, the feedback provided to an LLM agent after code execution can vary substantially. Factors such as the execution environment, the expertise of the feedback provider, and the effort invested in analyzing and writing feedback to the code all influence the form and quality of the feedback. Among these, the provider's expertise and the depth of analysis play the most significant roles in determining how informative the feedback is. To systematically evaluate LLM agents under varying feedback conditions, we design a five-level feedback hierarchy, where each level provides progressively more guidance:

- **Level 0:** Minimal feedback. The agent is only informed that code execution failed and that the execution result log is available.
- **Level 1:** Execution result plus a high-level error description that briefly characterizes the failure.
- **Level 2:** In addition to Level 1, an explanation of why the error occurred, offering diagnostic insight.
- **Level 3:** Extends Level 2 by providing natural language guidance that specifies how the implementation should be corrected. The feedback outlines the appropriate algorithmic modifications or design choices needed to align the code with the canonical implementation, without supplying any executable code.
- **Level 4:** The most detailed setting. Beyond Level 3, the exact ground-truth code snippet is provided. This level isolates the model's ability to accurately integrate a correct solution into the existing repository, serving as an upper-bound condition for evaluating instruction adherence and repository-level integration.

Feedback Generation. We employ `GPT-o4-mini` to simulate an expert and generate feedback to ensure the benchmark's reproducibility and scalability. After each code-generation round, the feedback model provides concise, actionable feedback. The feedback is conditioned on code execution results, including test outcomes and error logs. It also leverages the canonical code and annotated comments to clarify functionality and verify alignment with the paper's description. Using an LLM to produce feedback in this loop preserves the iterative nature of real research workflows while ensuring that feedback is standardized and reproducible across runs. We chose `GPT-o4-mini` because of its cost efficiency and high-quality feedback, as discussed in Appendix F.

Evaluation Metrics. We assess LLM agents on RECODE-H by functional correctness and code similarity. Functional correctness is measured with test cases using Mean Reciprocal Rank (MRR), which assigns a score of $\frac{1}{k}$ based on the first turn k where correct code appears, and Recall@n, which is the proportion of tasks solved correctly within n turns. We also report the average proportion of test cases that pass to capture partial correctness. Code similarity is evaluated against canonical implementations using CodeBLEU (Ren et al., 2020), which combines lexical and structural signals, and CodeBERTScore (Zhou et al., 2023), which measures semantic similarity via embeddings.

4 RECODEAGENT

To evaluate our benchmark, we introduce a feedback-driven LLM code agent, ReCode Agent. The agent iteratively incorporates researcher feedback to generate or modify code, aiming to produce fully correct and executable implementations. ReCode Agent engages in multi-turn interactions with a researcher, refining its output until all tests pass or a predefined interaction limit is reached.

Agent Strategy. Our agent strategy follows the ReAct framework (Yao et al., 2023) and is organized into four stages. (1) Observation. It gathers the current repository state, execution logs from previously submitted code, and researcher feedback. (2) Reflection. It analyzes failures and gaps against the task specification, integrates feedback into actionable insights, and ensures consistency with repository constraints. (3) Planning. It formulates a concise, structured plan for the next step, specifying the goal, target files or spans, and the intended effect. (4) Action. It executes one operation from the predefined action space according to the plan. It should be noticed that, in our agent design, the use of "one-shot" refers to the human–agent interaction pattern rather than the agent's internal ReAct steps. Existing benchmarks support only a single human prompt, after which the agent operates autonomously. In RECODE, the agent engages in multi-turn human–agent collaboration, even though its internal reasoning–acting loop remains the same as standard ReAct.

Memory Management. To keep context length bounded under multi-round interaction, the agent maintains a memory similar to Reflexion (Shinn et al., 2023). We enforce a threshold on the number of recent memories to keep. When the memory exceeds the threshold, the agent compacts prior observations and actions into a concise summary that preserves the unresolved failures, design decisions, and generation context information. This memory compression promotes consistency across rounds while avoiding context bloat.

5 EXPERIMENTS

Experimental Setup. We evaluate ReCode Agent on RECODE-H using seven mainstream LLMs, spanning both reasoning and not-reasoning models. Specifically, we evaluated the GPT(OpenAI, 2025) family (`GPT-5, GPT-5-mini, GPT-5-nano`), the Gemini(Team et al., 2023) family (`Gemini-2.5-pro and Gemini-2.5-flash`), `Claude-Sonnet-4`(Anthropic, 2025) from Anthropic, and `DeepSeek-V3.1`(DeepSeek, 2025). Each model is assessed under multiple feedback conditions in a multi-turn setting to examine performance across varying levels of human interaction. For code generation, we fix the decoding temperature to 0 and top-p to 1, ensuring deterministic outputs. The evaluation for each task proceeds for up to 10 rounds of feedback generation interaction. Within a single interaction turn between the LLM agent and the simulated human, we limit the agent to have at most 3 actions before automatic submission. And the threshold number of memory the LLM agent keeps is set to 5.

5.1 OVERALL RESULTS AFTER 10 INTERACTION TURNS

Overview. As shown in the Table 2, the richer feedback consistently improves LLM performance. With minimal feedback, models achieve low pass rates and limited correctness. As feedback becomes more detailed, from Levels 1 to 3, both success rates and efficiency improve, reflected in higher MRR, Recall, and test case pass rates. At the most detailed level, all models show substantial gains, with `GPT-5` and `Deepseek-V3.1` benefiting the most.

Model size and capability play a clear role in performance. Within the GPT family, larger models consistently achieve higher scores across all feedback levels, showing that scaling up enhances the ability to utilize feedback effectively. However, this trend is less evident in Gemini and Claude than in small-sized GPT models. Despite their relatively large sizes, their improvements from richer feedback are modest compared to `GPT-5` or `Deepseek-V3.1`. We attribute this to a lower feedback adoption rate as these models appear less efficient at incorporating feedback signals into subsequent generations, as discussed in Section 5.4. As a result, their final performance lags behind other large models that adapt more quickly and fully to iterative guidance.

Non-linear gains across feedback levels. We observe the improvements across feedback levels in Table 2 are non-linear, with the largest performance boost often observed from Level 0 to Level 1. For example, most models nearly double their Recall and test case pass rates once minimal di-

Model	Feedback Level	Pass Rate			Similarity	
		MRR(%)	Recall(%)	Test Case(%)	CodeBLEU	CodeBERT
OpenAI GPT						
GPT-5-nano	Level 0	1.4	5.9	23.8	0.252	0.854
	Level 1	3.0	10.8(**+4.9**)	31.9(**+8.1**)	0.258	0.821
	Level 2	4.1	16.7(**+5.9**)	39.4(**+7.5**)	0.262	0.843
	Level 3	4.6	19.6(**+2.9**)	42.2(**+2.8**)	0.266	0.817
	Level 4	9.1	35.3(**+15.7**)	54.7(**+12.5**)	0.290	0.857
GPT-5-mini	Level 0	1.3	19.6	42.3	0.262	0.870
	Level 1	6.2	38.2(**+18.6**)	62.8(**+20.5**)	0.299	0.884
	Level 2	7.8	47.1(**+8.9**)	70.2(**+7.4**)	0.303	0.887
	Level 3	8.8	52.9(**+5.8**)	73.2(**+3.0**)	0.303	0.888
	Level 4	11.1	66.7(**+13.8**)	81.0(**+7.8**)	0.309	0.889
GPT-5	Level 0	6.0	29.4	53.7	0.287	0.879
	Level 1	7.5	45.1(**+15.7**)	68.8(**+15.1**)	0.294	0.884
	Level 2	9.3	55.9(**+10.8**)	77.4(**+8.6**)	0.302	0.885
	Level 3	10.6	63.7(**+7.8**)	83.7(**+6.3**)	0.304	0.886
	Level 4	11.9	71.6(**+7.9**)	84.3(**+0.6**)	0.317	0.888
Anthropic Claude						
Claude-sonnet-4	Level 0	5.2	14.7	35.1	0.287	0.885
	Level 1	6.0	24.5(**+9.8**)	48.5(**+13.4**)	0.303	0.881
	Level 2	7.5	33.3(**+8.8**)	51.8(**+3.3**)	0.300	0.888
	Level 3	7.7	33.3(**+0.0**)	53.5(**+1.7**)	0.312	0.890
	Level 4	11.4	48.0(**+14.7**)	64.4(**+1.7**)	0.335	0.892
Deepseek						
DeepSeek-V3.1	Level 0	5.1	10.8	30.7	0.292	0.886
	Level 1	8.9	26.5(**+15.7**)	47.2(**+16.5**)	0.292	0.890
	Level 2	12.1	43.1(**+16.6**)	60.6(**+13.4**)	0.301	0.891
	Level 3	14.1	49.0(**+5.9**)	57.6(**-3.0**)	0.308	0.892
	Level 4	21.0	70.6(**+21.6**)	77.3(**+19.7**)	0.341	0.897
Google Gemini						
Gemini-2.5-flash	Level 0	3.9	8.8	32.2	0.294	0.889
	Level 1	7.4	27.5(**+18.7**)	52.6(**+20.4**)	0.300	0.891
	Level 2	9.6	34.3(**+6.8**)	58.7(**+6.1**)	0.309	0.893
	Level 3	12.0	47.1(**+12.8**)	70.7(**+12.0**)	0.311	0.893
	Level 4	14.2	58.8(**+11.7**)	77.6(**+6.9**)	0.348	0.899
Gemini-2.5-pro	Level 0	4.3	12.7	35.5	0.265	0.876
	Level 1	6.1	16.7(**+4.0**)	40.1(**+4.6**)	0.278	0.880
	Level 2	9.6	37.3(**+20.6**)	57.6(**+17.5**)	0.285	0.883
	Level 3	10.4	37.3(**+0.0**)	58.0(**+0.4**)	0.291	0.874
	Level 4	17.6	58.8(**+21.5**)	70.6(**+12.6**)	0.331	0.888

Table 2: This table reports performance of diverse LLMs on the RECODE benchmark over 10 rounds of interaction. Metrics include Mean Reciprocal Rank (MRR), Recall, average test case pass rate, and code similarity scores (CodeBLEU and CodeBERTScore). Stronger feedback consistently improves functional correctness and alignment with ground-truth implementations, with higher feedback levels leading to more effective error correction and refinement

agnostic information is provided, indicating that even shallow guidance strongly accelerates error correction. In contrast, the gains from Level 2 to Level 3 and from Level 3 to Level 4 are more moderate, suggesting diminishing returns as feedback becomes increasingly detailed. Nevertheless, the highest level of feedback, which provides explicit code snippets, still delivers substantial improvements in functional correctness, especially for stronger models like `GPT-5` and `Deepseek-V3.1`. This pattern demonstrates that while structured feedback is highly valuable, the marginal benefits taper off as it approaches full supervision.

Differences across model families. The results also reveal notable differences across model families. The GPT family shows consistent and significant improvements as feedback becomes richer, with `GPT-5` and `GPT-5-mini` maintaining high scores across all metrics. In contrast, `Claude-Sonnet-4` shows only moderate gains, plateauing earlier and failing to fully utilize the detailed feedback. The Gemini family presents a split. `Gemini-2.5-flash` perform better than `Gemini-2.5-pro` across all levels. while `Gemini-2.5-pro` shows competitive results at higher feedback levels but still lags behind `GPT-5` and `Deepseek-V3.1`. Notably, `Deepseek-V3.1` demonstrates the largest relative improvement from Level 0 to Level 4, indicating a high sensitivity to feedback and strong adaptability in multi-turn interactions. These differ-

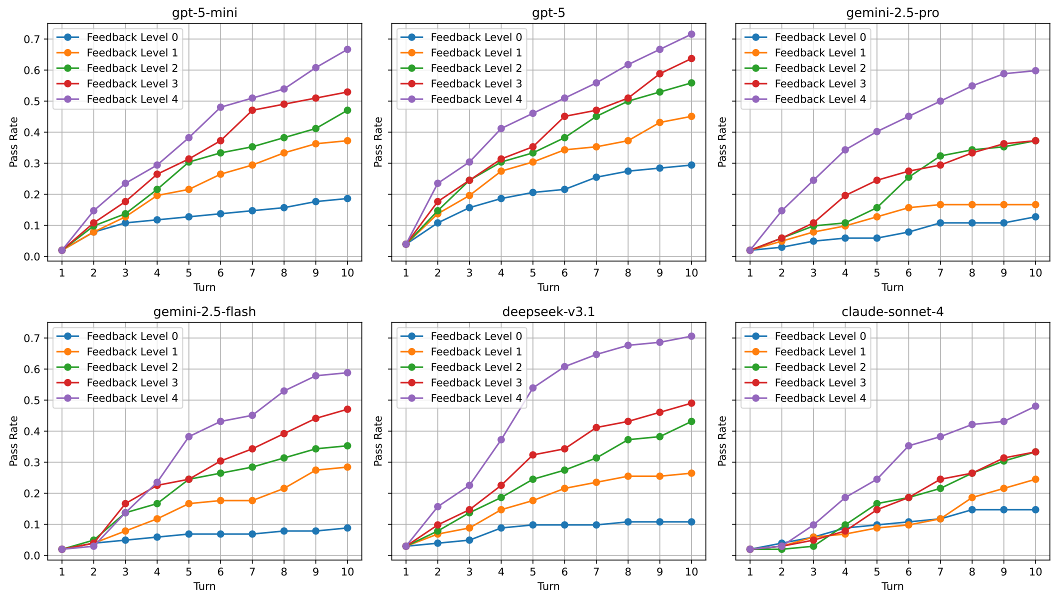


Figure 2: Pass rate trajectories across interaction turns under varying feedback levels. Richer feedback consistently boosts model performance, with the largest gains appearing in early turns. Stronger models like `GPT-5` and `Deepseek-V3.1` adapt more effectively, while `Gemini-2.5-flash` and `Claude-Sonnet-4` plateau earlier.

ences suggest that, beyond model size, the architecture and training approach of each family play a crucial role in determining how effectively models can incorporate iterative feedback into code generation.

Recall and MRR improvements align. A consistent trend across Table 2 is that Recall and MRR improve as feedback levels increase. `GPT-5`'s Recall rises steadily from 0.294 at Level 0 to 0.716 at Level 4, and its MRR improves from 0.060 to 0.119 over the same range.

5.2 Performance Dynamics Across Interaction Turns

Figure 2 illustrates the test case pass rate with the increase of interactive turns. The trajectories clearly show that richer feedback not only improves the final success rate but also accelerates convergence in early turns. Both `GPT-5` and `DeepSeek-V3.1` rapidly increase their pass rates within the first 3–4 turns when provided with Level 3 or Level 4 feedback, whereas with Level 0 feedback their improvement remains gradual and plateaus at much lower levels. In addition, the first-turn results shown in Figure 2 also serve as our single-turn baseline. In this initial turn, the model receives only the task prompt without any interactive feedback, corresponding to the highlighted single-turn setting. As the figure indicates, subsequent turns that incorporate richer feedback yield substantial improvements over this baseline. After 10 turns, all models demonstrate markedly higher pass rates, confirming that multi-turn interaction provides a significant uplift over a single-turn scenario.

In contrast, smaller or weaker models exhibit slower gains and limited sensitivity to feedback richness, resulting in lower overall performance. `Gemini-2.5-pro` and `Claude-Sonnet-4` exhibit unstable performance gaps between feedback levels as interaction turns increase, which aligns with the inconsistent feedback adoption rates discussed in Section 5.4 and Appendix J. Among them, `Claude-Sonnet-4` shows the weakest performance trajectory across turns: the differences between feedback Levels 1–3 are far less pronounced than in other models, indicating difficulty in effectively leveraging moderate feedback.

5.3 Error Analysis

To better understand the role of feedback in guiding multi-turn code generation, we conducted a fine-grained analysis of the errors encountered during the benchmark and the corrective signals associated with them. Our analysis revealed that errors can be systematically categorized into four types based on their root causes, each reflecting a different source of failure in the generation process.

Type 1: Code Syntax and Runtime Errors. This category encompasses fundamental programming issues that are independent of high-level algorithm design. Typical cases include syntax violations, runtime failures such as type mismatches or out-of-bounds indices, and basic logic flaws such as incorrect loop termination or variable misuse.

Type 2:Paper and Instruction Misunderstanding Errors. These errors arise when the generated code diverges from the intended method or fails to meet the benchmark requirements. Common manifestations include incorrect implementations of mathematical formulas, omission or misapplication of algorithmic steps, and mismatches in input/output formats or tensor shapes. They reflect a misalignment between the model's output and the research description it is expected to reproduce.

Type 3: Missing Knowledge and Context Errors. Some failures result from incomplete domain knowledge or implicit assumptions that are not explicitly stated in the task description. Examples include neglecting to use standard library functions, misunderstanding technical terminology, or overlooking repository-specific conventions. These errors highlight the model's difficulty in bridging gaps between textual instructions and domain practices.

Type 4: Repository Integration Errors. Finally, certain errors are caused not by the algorithm itself, but by poor integration with the surrounding codebase. These include reimplementing existing helper functions, misusing predefined modules or classes, or disregarding established coding conventions. Such errors limit the usability of generated code within the broader repository and emphasize the importance of context-aware generation.

LLM	**Type1(%)**	**Type2(%)**	**Type3(%)**	**Type4(%)**
Gpt-5	11.35	34.04	50.25	4.36
Gpt-5-mini	11.53	27.06	55.22	6.19
Gpt-5-nano	20.82	37.32	34.95	6.90
Gemini-2.5-pro	14.94	39.91	37.34	7.82
Gemini-2.5-flash	16.16	26.15	49.41	8.28
Deepseek-chat	20.60	31.64	40.30	7.46
Claude-sonnet-4	26.45	32.26	33.75	7.54

Table 3: Error type distribution across models on the RECODE-H benchmark.

In this experiment, we employ `GPT-5` to classify the reasons for errors. To ensure the reliability of the classification, we randomly sampled 100 cases and verified them with human annotators, achieving 98% agreement with `GPT-5`'s predictions, providing strong evidence of classification accuracy. As shown in Table 3, the majority of failures are attributed to higher-level semantic issues rather than low-level coding mistakes. Specifically, paper and instruction misunderstanding errors(Type 2), missing knowledge and context errors(Type 3) dominate across all models, whereas syntax and runtime errors(Type 1) occur less frequently, and repository integration errors(Type 4) are the least common. This distribution shows that modern LLMs have largely overcome basic coding challenges. However, they still struggle to align implementations faithfully with research descriptions and to bridge implicit domain knowledge. In addition, occasional but impactful gaps in repository awareness remain. A more detailed discussion of error type patterns and their implications is provided in Appendix I.

5.4 Feedback Adoption

We analyze how often models adopt the provided feedback and whether adoption results in a correct fix. `GPT-5` is employed as a classifier to determine whether the feedback is incorporated into the revised code and whether the targeted error is resolved. Our evaluation shows that both adoption and fix rates vary substantially across models, guidance levels, and feedback types. Detailed Table 10 and Table 11 supporting these findings are provided in Appendix J.

Adoption as a necessary pathway. Across all models and settings, nearly every successfully corrected error is one where the model explicitly adopted the provided feedback. Cases where errors were fixed without adoption are exceedingly rare, underscoring that feedback-driven improvement is the main driver of repair.

Effect of guidance level on adoption. Stronger feedback guidance increases the likelihood of adoption overall, but the magnitude of this increase differs substantially across models. Models that exhibit significant improvement in pass rates as guidance level rises, such as `GPT-5`, `GPT-5-mini`, and `DeepSeek-V3.1`, also show clear gains in feedback adoption. For instance, `GPT-5` adoption rises from 80.2% at Level-1 to 90.1% at Level-4, with `DeepSeek-V3.1` and `GPT-5-mini` following similar upward trajectories. This pattern suggests that the models most capable of leveraging feedback effectively are increasingly receptive to guidance as it becomes more explicit.

Declining or fluctuating adoption among weaker improvers. By contrast, models that show low pass rate under stronger guidance, such as `Gemini-2.5-pro` and `Claude-Sonnet-4`, exhibit a different pattern of adoption. For instance, the adoption rate of `Claude-Sonnet-4` decreases as the feedback guidance level increases, while the adoption rate of `Gemini-2.5-pro` fluctuates around 70%. These feedback adoption patterns align with their weaker performance in the benchmark.

Simple and highly-specific feedback is adopted most. Models exhibit the highest adoption rates for feedback targeting Code Correctness and Repository Integration errors. For example, `DeepSeek-chat` adopts nearly 80% of syntax-related feedback. In contrast, adoption rates of feedback for implementation alignment error (Type 2) are consistently lower across models, with `GPT-5-nano` showing the lowest rate at 56.1%. This pattern suggests that models struggle to address subtle logical errors that require a deeper understanding of the research method's intent.

5.5 Discussion and Insights

Our results highlight that the main barrier in research code generation is semantic fidelity. Models most often fail by misinterpreting methodological intent or missing domain knowledge, rather than through syntactic errors. Feedback effects are also highly non-linear. The minimal diagnostic signals yield the largest improvements, while richer natural-language guidance yields diminishing returns until explicit code snippets are supplied. Furthermore, feedback utilization varies sharply across models, indicating that effective incorporation of iterative guidance is a model-dependent capability. These observations suggest that future research should prioritize stronger semantic grounding and more stable multi-turn refinement. In practice, concise execution-level feedback is most effective, while human effort is best spent addressing conceptual misalignments.

6 Conclusion

We introduce RECODE-H, a benchmark for evaluating LLM-based research agents in realistic scientific workflows with feedback-driven code implementation. Unlike prior work focused on end-to-end pipelines or one-shot generation, RECODE-H emphasizes interactive evaluation, reflecting core aspects of real research practice. Our experiments reveal three findings: (1) scientific grounding, not coding, is the primary bottleneck, as models struggle with methodological interpretation and implicit domain knowledge; (2) feedback yields strongly non-linear gains, with minimal signals unlocking substantial improvements; and (3) feedback receptivity is a distinct capability, with stronger models benefiting from richer guidance while weaker ones plateau. Future work includes extending the benchmark to broader research stages, multi-agent settings, and human-in-the-loop feedback. Overall, effective research agents will require not only coding ability but also adaptive reasoning and sustained interaction with complex research environments.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) under Grants III-2106758 and POSE-2346158.

Yangning is supported by the Basic Research Fund of Shenzhen City (Grant No. JCYJ20240813112009013).

REFERENCES

Silas Alberti, Kenan Hasanaliyev, Manav Shah, and Stefano Ermon. Data unlearning in diffusion models. *arXiv preprint arXiv:2503.01034*, 2025.

Anthropic. Introducing claude 4. `https://www.anthropic.com/news/claude-4`, May 22 2025. Accessed: 2025-09-20.

Abolfazl Asudeh, Nima Shahbazi, and Stavros Sintos. A fair and memory/time-efficient hashmap. *arXiv preprint arXiv:2307.11355*, 2023.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Kumar Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, A. Farahani, Siddharth Jain, Robert Giaquinto, Haifeng Qian, M. Ramanathan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, Sudipta Sengupta, D. Roth, and Bing Xiang. Multi-lingual evaluation of code generation models. *ArXiv*, abs/2210.14868, 2022. URL `https://arxiv.org/pdf/2210.14868.pdf`.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

L Lao Beyer, Tianhong Li, Xinlei Chen, Sertac Karaman, and Kaiming He. Highly compressed tokenizer can generate without training. *arXiv preprint arXiv:2506.08257*, 2025.

Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. Lightgcl: Simple yet effective graph contrastive learning for recommendation. *arXiv preprint arXiv:2302.08191*, 2023.

Jialun Cao, Yuk-Kit Chan, Zixuan Ling, Wenxuan Wang, Shuqing Li, Mingwei Liu, Ruixi Qiao, Yuting Han, Chaozheng Wang, Boxi Yu, et al. How should we build a benchmark? revisiting 274 code-related benchmarks for llms. *arXiv preprint arXiv:2501.10711*, 2025.

Zhiwei Cao, Qian Cao, Yu Lu, Ningxin Peng, Luyang Huang, Shanbo Cheng, and Jinsong Su. Retaining key information under high compression ratios: Query-guided compressor for llms. *arXiv preprint arXiv:2406.02376*, 2024.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Chen Chen, Ruizhe Li, Yuchen Hu, Sabato Marco Siniscalchi, Pin-Yu Chen, Ensiong Chng, and Chao-Han Huck Yang. It's never too late: Fusing acoustic information into large language models for automatic speech recognition. *arXiv preprint arXiv:2402.05457*, 2024a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023a.

Yankai Chen, Yixiang Fang, Yifei Zhang, and Irwin King. Bipartite graph convolutional hashing for effective and efficient top-n search in hamming space. In *Proceedings of the ACM Web Conference 2023*, pp. 3164–3172, 2023b.

Yankai Chen, Yixiang Fang, Yifei Zhang, Chenhao Ma, Yang Hong, and Irwin King. Towards effective top-n hamming search via bipartite graph contrastive hashing. *IEEE Trans. on Knowl. and Data Eng.*, 36(12):9418–9432, December 2024b. ISSN 1041-4347. doi: 10.1109/TKDE.2024.3425891. URL `https://doi.org/10.1109/TKDE.2024.3425891`.

Yankai Chen, Taotao Wang, Yixiang Fang, and Yunyu Xiao. Semi-supervised node importance estimation with informative distribution modeling for uncertainty regularization. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pp. 3108–3118, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400712746. doi: 10.1145/3696410.3714591. URL https://doi.org/10.1145/3696410.3714591.

Yankai Chen, Xinni Zhang, Yifei Zhang, Yangning Li, Henry Peng Zou, Chunyu Miao, Weizhi Zhang, Xue Liu, and Philip S Yu. Embracing trustworthy brain-agent collaboration as paradigm extension for intelligent assistive technologies. *arXiv preprint arXiv:2510.22095*, 2025b.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. *arXiv preprint arXiv:2410.05080*, 2024c.

Ziqiang Cui, Xing Tang, Yang Qiao, Bowei He, Liang Chen, Xiuqiang He, and Chen Ma. Treatment-aware hyperbolic representation learning for causal effect estimation with social networks. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*, pp. 289–297. SIAM, 2024a.

Ziqiang Cui, Haolun Wu, Bowei He, Ji Cheng, and Chen Ma. Diffusion-based contrastive learning for sequential recommendation. *arXiv preprint arXiv:2405.09369*, 2024b.

Yizhou Dang, Jiahui Zhang, Yuting Liu, Enneng Yang, Yuliang Liang, Guibing Guo, Jianzhe Zhao, and Xingwei Wang. Augmenting sequential recommendation with balanced relevance and diversity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 11563–11571, 2025.

Inc. DeepSeek. Deepseek-v3.1 release. https://api-docs.deepseek.com/news/news250821, August 21 2025. Accessed: 2025-09-20.

Aaron Defazio, Xingyu Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled. *Advances in Neural Information Processing Systems*, 37: 9974–10007, 2024.

Mohsen Dehghankar, Mahdi Erfanian, and Abolfazl Asudeh. An efficient matrix multiplication algorithm for accelerating inference in binary and ternary neural networks. *arXiv preprint arXiv:2411.06360*, 2024.

Mohsen Dehghankar, Rahul Raychaudhury, Stavros Sintos, and Abolfazl Asudeh. Fair set cover. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 189–200, 2025.

Karan Desai, Maximilian Nickel, Tanmay Rajpurohit, Justin Johnson, and Shanmukha Ramakrishna Vedantam. Hyperbolic image-text representations. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 7694–7731. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/desai23a.html.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, M. K. Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *ArXiv*, abs/2310.11248, 2023. URL https://arxiv.org/pdf/2310.11248.pdf.

Xiangyu Dong, Xingyi Zhang, Yanni Sun, Lei Chen, Mingxuan Yuan, and Sibo Wang. Smoothgnn: Smoothing-aware gnn for unsupervised node anomaly detection. *arXiv preprint arXiv:2405.17525*, 2024.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, et al. Loramoe: Alleviate world knowledge forgetting in large language models via moe-style plugin. *arXiv preprint arXiv:2312.09979*, 2023.

Tianyu Du, Luca Melis, and Ting Wang. Remasker: Imputing tabular data with masked autoencoding. *arXiv preprint arXiv:2309.13793*, 2023.

Benoit Dufumier, Javiera Castillo-Navarro, Devis Tuia, and Jean-Philippe Thiran. What to align in multimodal contrastive learning? *arXiv preprint arXiv:2409.07402*, 2024.

Nicholas Edwards, Yukyung Lee, Yujun Audrey Mao, Yulu Qin, Sebastian Schuster, and Najoung Kim. Rexbench: Can coding agents autonomously implement ai research extensions? *arXiv preprint arXiv:2506.22598*, 2025.

Lan Feng, Fan Nie, Yuejiang Liu, and Alexandre Alahi. Tarot: Targeted data selection via optimal transport. *arXiv preprint arXiv:2412.00420*, 2024.

Soham Gadgil, Ian Covert, and Su-In Lee. Estimating conditional mutual information for dynamic feature selection. *arXiv preprint arXiv:2306.03301*, 2023.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*, 2024.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36:19622–19635, 2023.

Jialong Guo, Xinghao Chen, Yehui Tang, and Yunhe Wang. Slab: Efficient transformers with simplified linear attention and progressive re-parameterized batch normalization. *arXiv preprint arXiv:2405.11582*, 2024.

Hojae Han, Seung-won Hwang, Rajhans Samdani, and Yuxiong He. Convcodeworld: Benchmarking conversational code generation in reproducible feedback environments. *arXiv preprint arXiv:2502.19852*, 2025.

Bowei He, Yankai Chen, Xiaokun Zhang, Linghe Kong, Philip S. Yu, Xue Liu, and Chen Ma. Pedagogically inspired data synthesis for language model knowledge distillation. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, D. Song, and J. Steinhardt. Measuring coding challenge competence with apps. *ArXiv*, abs/2105.09938, 2021. URL `https://arxiv.org/pdf/2105.09938.pdf`.

Guoqing Hu, An Zhang, Shuo Liu, Zhibo Cai, Xun Yang, and Xiang Wang. Alphafuse: Learn id embeddings for sequential recommendation in null space of language embeddings. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1614–1623, 2025.

Jiaxi Hu, Yuehong Hu, Wei Chen, Ming Jin, Shirui Pan, Qingsong Wen, and Yuxuan Liang. Attractor memory for long-term time series forecasting: A chaos perspective. *Advances in Neural Information Processing Systems*, 37:20786–20818, 2024.

Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T. Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. Researchcodebench: Benchmarking llms on implementing novel machine learning research code. *ArXiv*, abs/2506.02314, 2025. URL `https://api.semanticscholar.org/CorpusId:279119993`.

Baizhou Huang, Shuai Lu, Xiaojun Wan, and Nan Duan. Enhancing large language models in coding through multi-perspective self-consistency. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1429–1450, 2024.

Wei-Chieh Huang and Cornelia Caragea. Madiave: Multi-agent debate for implicit attribute value extraction. *arXiv preprint arXiv:2510.05611*, 2025.

Wei-Chieh Huang, Henry Peng Zou, Yaozu Wu, Dongyuan Li, Yankai Chen, Weizhi Zhang, Yangning Li, Angelo Zangari, Jizhou Guo, Chunyu Miao, et al. Deepresearchguard: Deep research with open-domain evaluation and multi-stage guardrails for safety. *arXiv preprint arXiv:2510.10994*, 2025.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. Rethinking memory mechanisms of foundation agents in the second half. *arXiv preprint arXiv:2602.06052*, 2026.

Guillaume Jaume, Anurag Vaidya, Richard J Chen, Drew FK Williamson, Paul Pu Liang, and Faisal Mahmood. Modeling dense multimodal interactions between biological pathways and histology for survival prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11579–11590, 2024.

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? *arXiv preprint arXiv:2310.06770*, 2023.

Tommie Kerssies, Niccolo Cavagnero, Alexander Hermans, Narges Norouzi, Giuseppe Averta, Bastian Leibe, Gijs Dubbelman, and Daan de Geus. Your vit is secretly an image segmentation model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 25303–25313, 2025.

Patrick Tser Jern Kon, Jiachen Liu, Xinyi Zhu, Qiuyi Ding, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, et al. Exp-bench: Can ai conduct ai research experiments? *arXiv preprint arXiv:2505.24785*, 2025.

Dilfira Kudrat, Zongxia Xie, Yanru Sun, Tianyu Jia, and Qinghua Hu. Patch-wise structural loss for time series forecasting. *arXiv preprint arXiv:2503.00877*, 2025.

Chengpeng Li, Zhengyang Tang, Ziniu Li, Mingfeng Xue, Keqin Bao, Tian Ding, Ruoyu Sun, Benyou Wang, Xiang Wang, Junyang Lin, and Dayiheng Liu. Cort: Code-integrated reasoning within thinking. *ArXiv*, abs/2506.09820, 2025a. URL `https://api.semanticscholar.org/CorpusId:279305850`.

Jia Li, Ge Li, Xuanming Zhang, Yihong Dong, and Zhi Jin. Evocodebench: An evolving code generation benchmark aligned with real-world code repositories. *arXiv preprint arXiv:2404.00599*, 2024a.

Jia Li, Ge Li, Yunfei Zhao, Yongmin Li, Huanyu Liu, Hao Zhu, Lecheng Wang, Kaibo Liu, Zheng Fang, Lanshen Wang, et al. Deveval: A manually-annotated code generation benchmark aligned with real-world code repositories. *arXiv preprint arXiv:2405.19856*, 2024b.

Jiangmeng Li, Yifan Jin, Hang Gao, Wenwen Qiang, Changwen Zheng, and Fuchun Sun. Hierarchical topology isomorphism expertise embedded graph contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13518–13527, 2024c.

Ryan Li, Yanzhe Zhang, and Diyi Yang. Sketch2code: Evaluating vision-language models for interactive web design prototyping. In *North American Chapter of the Association for Computational Linguistics*, 2024d. URL `https://api.semanticscholar.org/CorpusId:273507760`.

Wen-Zhi Li, Chang-Dong Wang, Hui Xiong, and Jian-Huang Lai. Homogcl: Rethinking homophily in graph contrastive learning. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 1341–1352, 2023.

Yangning Li, Weizhi Zhang, Yuyao Yang, Wei-Chieh Huang, Yaozu Wu, Junyu Luo, Yuanchen Bei, Henry Peng Zou, Xiao Luo, Yusheng Zhao, et al. A survey of rag-reasoning systems in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12120–12145, 2025b.

Zhixun Li, Yushun Dong, Qiang Liu, and Jeffrey Xu Yu. Rethinking fair graph neural networks from re-balancing. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 1736–1745, 2024e.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhijiang Guo, Le Song, and Cheng-Lin Liu. From system 1 to system 2: A survey of reasoning large language models. *ArXiv*, abs/2502.17419, 2025c. URL `https://api.semanticscholar.org/CorpusId:276575321`.

Zongxia Li, Ishani Mondal, Huy Nghiem, Yijun Liang, and Jordan Lee Boyd-Graber. Pedants: Cheap but effective and interpretable answer equivalence. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 9373–9398, 2024f.

Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 365–374, 2024a.

Yiqing Lin, Jianheng Tang, Chenyi Zi, H Vicky Zhao, Yuan Yao, and Jia Li. Unigad: Unifying multi-level graph anomaly detection. *Advances in neural information processing systems*, 37: 136120–136148, 2024b.

Chongming Liu, Songting Li, and Douglas Zhou. Dendritic integration based quadratic neural networks outperform traditional aritificial ones. *arXiv preprint arXiv:2307.13609*, 2023a.

Meng Liu, Yue Liu, Ke Liang, Wenxuan Tu, Siwei Wang, Sihang Zhou, and Xinwang Liu. Deep temporal graph clustering. *arXiv preprint arXiv:2305.10738*, 2023b.

Pei Liu, Luping Ji, Jiaxiang Gou, Bo Fu, and Mao Ye. Interpretable vision-language survival analysis with ordinal inductive bias for computational pathology. *arXiv preprint arXiv:2409.09369*, 2024a.

Qidong Liu, Xian Wu, Yejing Wang, Zijian Zhang, Feng Tian, Yefeng Zheng, and Xiangyu Zhao. Llm-esr: Large language models enhancement for long-tailed sequential recommendation. *Advances in Neural Information Processing Systems*, 37:26701–26727, 2024b.

Shuo Liu, An Zhang, Guoqing Hu, Hong Qian, and Tat-seng Chua. Preference diffusion for recommendation. *arXiv preprint arXiv:2410.13117*, 2024c.

Yixin Liu, Shiyuan Li, Yu Zheng, Qingfeng Chen, Chengqi Zhang, and Shirui Pan. Arc: A generalist graph anomaly detector with in-context learning. *Advances in Neural Information Processing Systems*, 37:50772–50804, 2024d.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023c.

Yue Liu, Shihao Zhu, Jun Xia, Yingwei Ma, Jian Ma, Xinwang Liu, Shengju Yu, Kejun Zhang, and Wenliang Zhong. End-to-end learnable clustering for intent learning in recommendation. *Advances in Neural Information Processing Systems*, 37:5913–5949, 2024e.

Jiecheng Lu, Yan Sun, and Shihao Yang. In-context time series predictor. *arXiv preprint arXiv:2405.14982*, 2024.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Odelia Melamed, Gilad Yehudai, and Adi Shamir. Malt powers up adversarial attacks. *Advances in Neural Information Processing Systems*, 37:44366–44392, 2024.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235, 2024.

Jiacheng Miao and Qiongshi Lu. Task-agnostic machine-learning-assisted inference. *Advances in Neural Information Processing Systems*, 37:106162–106189, 2024.

Christopher Mohri and Tatsunori Hashimoto. Language models with conformal factuality guarantees. *arXiv preprint arXiv:2402.10978*, 2024.

Minh Nhat Nguyen, Andrew Baker, Clement Neo, Allen Roush, Andreas Kirsch, and Ravid Shwartz-Ziv. Turning up the heat: Min-p sampling for creative and coherent llm outputs. *arXiv preprint arXiv:2407.01082*, 2024.

Y Nie. A time series is worth 64words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

OpenAI. Gpt-5 system card. Technical report, OpenAI, 2025. URL `https://cdn.openai.com/gpt-5-system-card.pdf`. Accessed: 2025-09-20.

Harshith Padigela, Chintan Shah, and Dinkar Juyal. Ml-dev-bench: Comparative analysis of ai agents on ml development workflows. *arXiv preprint arXiv:2502.00964*, 2025.

Ziheng Qin, Kai Wang, Zangwei Zheng, Jianyang Gu, Xiangyu Peng, Zhaopan Xu, Daquan Zhou, Lei Shang, Baigui Sun, Xuansong Xie, et al. Infobatch: Lossless training speed up by unbiased dynamic data pruning. *arXiv preprint arXiv:2303.04947*, 2023.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Maxon Rubin-Toles, Maya Gambhir, Keshav Ramji, Aaron Roth, and Surbhi Goel. Conformal language model reasoning with coherent factuality. *arXiv preprint arXiv:2505.17126*, 2025.

Philipp Seidl, Andreu Vall, Sepp Hochreiter, and Günter Klambauer. Enhancing activity prediction models in drug discovery with the ability to understand human language. In *International Conference on Machine Learning*, pp. 30458–30490. PMLR, 2023.

Yunfan Shao, Linyang Li, Zhaoye Fei, Hang Yan, Dahua Lin, and Xipeng Qiu. Balanced data sampling for language model training with clustering. *arXiv preprint arXiv:2402.14526*, 2024.

Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. Tabdiff: a mixed-type diffusion model for tabular data generation. *arXiv preprint arXiv:2410.20626*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can llms generate novel research ideas? a large-scale human study with 100+ nlp researchers. *arXiv preprint arXiv:2409.04109*, 2024.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can LLMs generate novel research ideas? a large-scale human study with 100+ NLP researchers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=M23dTGWCZy`.

Andrew H Song, Richard J Chen, Guillaume Jaume, Anurag J Vaidya, Alexander S Baras, and Faisal Mahmood. Multimodal prototyping for cancer survival prediction. *arXiv preprint arXiv:2407.00224*, 2024.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.

Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhe-Wei Shen, Baocai Chen, Lu Chen, and Kai Yu. Scieval: A multi-level large language model evaluation benchmark for scientific research. *ArXiv*, abs/2308.13149, 2023. URL `https://api.semanticscholar.org/CorpusId:261214653`.

Sophia Sun and Rose Yu. Copula conformal prediction for multi-step time series forecasting. *arXiv preprint arXiv:2212.03281*, 2022.

Yao Tang, Zhihui Xie, Zichuan Lin, Deheng Ye, and Shuai Li. Learning versatile skills with curriculum masking. *Advances in Neural Information Processing Systems*, 37:65562–65582, 2024.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, N. Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, A. Lupidi, Andrei Lupu, R. Raileanu, Kelvin Niu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H. Miller, Abhishek Charnalia, Derek Dunfield, Carole-Jean Wu, Pontus Stenetorp, Nicola Cancedda, J. Foerster, and Yoram Bachrach. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *ArXiv*, abs/2507.02554, 2025. URL `https://api.semanticscholar.org/CorpusId:280148761`.

Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y Zhang, and Jun Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. *arXiv preprint arXiv:2405.14616*, 2024a.

Xiangsen Wang, Haoran Xu, Yinan Zheng, and Xianyuan Zhan. Offline multi-agent reinforcement learning with implicit global-to-local value regularization. *Advances in Neural Information Processing Systems*, 36:52413–52429, 2023a.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*, 2023b.

Yidong Wang, Qi Guo, Wenjin Yao, Hongbo Zhang, Xin Zhang, Zhen Wu, Meishan Zhang, Xinyu Dai, Qingsong Wen, Wei Ye, et al. Autosurvey: Large language models can automatically write surveys. *Advances in neural information processing systems*, 37:115119–115145, 2024b.

Jeremy Wayland, Corinna Coupette, and Bastian Rieck. Mapping the multiverse of latent representations. *arXiv preprint arXiv:2402.01514*, 2024.

Tiansheng Wen, Yifei Wang, Zequn Zeng, Zhong Peng, Yudi Su, Xinyang Liu, Bo Chen, Hongwei Liu, Stefanie Jegelka, and Chenyu You. Beyond matryoshka: Revisiting sparse coding for adaptive representation. *arXiv preprint arXiv:2503.01776*, 2025.

Yixuan Weng, Minjun Zhu, Guangsheng Bao, Hongbo Zhang, Jindong Wang, Yue Zhang, and Linyi Yang. Cycleresearcher: Improving automated research via automated review. *ArXiv*, abs/2411.00816, 2024. URL `https://api.semanticscholar.org/CorpusId:273811997`.

Qitian Wu, Fan Nie, Chenxiao Yang, and Junchi Yan. Learning divergence fields for shift-robust graph representations. *arXiv preprint arXiv:2406.04963*, 2024a.

Qitian Wu, Chenxiao Yang, Kaipeng Zeng, and Michael M. Bronstein. Supercharging graph transformers with advective diffusion. In *Forty-second International Conference on Machine Learning*, 2025a. URL `https://openreview.net/forum?id=MaOYl3P84E`.

Shirley Wu, Kaidi Cao, Bruno Ribeiro, James Zou, and Jure Leskovec. Graphmetro: Mitigating complex graph distribution shifts via mixture of aligned experts. *arXiv preprint arXiv:2312.04693*, 2023.

Yaozu Wu, Dongyuan Li, Yankai Chen, Renhe Jiang, Henry Peng Zou, Wei-Chieh Huang, Yangning Li, Liancheng Fang, Zhen Wang, and Philip S Yu. Multi-agent autonomous driving systems with large language models: A survey of recent advances, resources, and future directions. *Findings of the Association for Computational Linguistics: EMNLP*, 2025, 2025b.

Zhaomin Wu, Junyi Hou, Yiqun Diao, and Bingsheng He. Federated transformer: Multi-party vertical federated learning on practical fuzzily linked data. *Advances in Neural Information Processing Systems*, 37:45791–45818, 2024b.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. Scireplicate-bench: Benchmarking llms in agent-driven algorithmic reproduction from research papers. *arXiv preprint arXiv:2504.00255*, 2025.

Teng Xiao, Huaisheng Zhu, Zhengyu Chen, and Suhang Wang. Simple and asymmetric graph contrastive learning without augmentations. *Advances in neural information processing systems*, 36:16129–16152, 2023.

Conghao Xiong, Hao Chen, Hao Zheng, Dong Wei, Yefeng Zheng, Joseph JY Sung, and Irwin King. Mome: Mixture of multimodal experts for cancer survival prediction. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 318–328. Springer, 2024.

Yanchen Xu, Siqi Huang, Hongyuan Zhang, and Xuelong Li. Why does dropping edges usually outperform adding edges in graph contrastive learning? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 21824–21832, 2025.

Yingxue Xu and Hao Chen. Multimodal optimal transport-based co-attention transformer with global structure consistency for survival prediction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 21241–21251, 2023.

Zhijian Xu, Ailing Zeng, and Qiang Xu. Fits: Modeling time series with $10k$ parameters. *arXiv preprint arXiv:2307.03756*, 2023.

Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, George Michalopoulos, Yue Zhang, et al. Lmr-bench: Evaluating llm agent's ability on reproducing language modeling research. *arXiv preprint arXiv:2506.17335*, 2025.

Hanchen Yang, Jiaqi Wang, Jiannong Cao, Wengen Li, Jialun Zheng, Yangning Li, Chunyu Miao, Jihong Guan, Shuigeng Zhou, and Philip S Yu. Okg-llm: Aligning ocean knowledge graph with observation data via llms for global sea surface temperature prediction. *arXiv preprint arXiv:2508.00933*, 2025.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854, 2023.

Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 15703–15712, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. Differential transformer. *arXiv preprint arXiv:2410.05258*, 2024.

Kun Yi, Qi Zhang, Wei Fan, Shoujin Wang, Pengyang Wang, Hui He, Ning An, Defu Lian, Longbing Cao, and Zhendong Niu. Frequency-domain mlps are more effective learners in time series forecasting. *Advances in Neural Information Processing Systems*, 36:76656–76679, 2023.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. *arXiv preprint arXiv:2410.06940*, 2024.

Yiwen Yuan, Zecheng Zhang, Xinwei He, Akihiro Nitta, Weihua Hu, Dong Wang, Manan Shah, Shenyang Huang, Blaž Stojanovič, Alan Krumholz, et al. Contextgnn: Beyond two-tower recommendation systems. *arXiv preprint arXiv:2411.19513*, 2024.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 11121–11128, 2023.

Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, et al. Wukong: Towards a scaling law for large-scale recommendation. *arXiv preprint arXiv:2403.02545*, 2024a.

Hanlei Zhang, Hua Xu, Fei Long, Xin Wang, and Kai Gao. Unsupervised multimodal clustering for semantics discovery in multimodal utterances. *arXiv preprint arXiv:2405.12775*, 2024b.

Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. *arXiv preprint arXiv:2310.09656*, 2023a.

Hengrui Zhang, Liancheng Fang, and Philip S Yu. Unleashing the potential of diffusion models for incomplete data imputation. *arXiv preprint arXiv:2405.20690*, 2024c.

Shichang Zhang, Jiani Zhang, Xiang Song, Soji Adeshina, Da Zheng, Christos Faloutsos, and Yizhou Sun. Page-link: Path-based graph neural network explanation for heterogeneous link prediction. In *Proceedings of the ACM web conference 2023*, pp. 3784–3793, 2023b.

Weizhi Zhang, Yangning Li, Yuanchen Bei, Junyu Luo, Guancheng Wan, Liangwei Yang, Chenxuan Xie, Yuyao Yang, Wei-Chieh Huang, Chunyu Miao, et al. From web search towards agentic deep research: Incentivizing search with reasoning agents. *arXiv preprint arXiv:2506.18959*, 2025a.

Yanbo Zhang, Sumeer A Khan, Adnan Mahmud, Huck Yang, Alexander Lavin, Michael Levin, Jeremy Frey, Jared Dunnmon, James Evans, Alan Bundy, et al. Advancing the scientific method with large language models: From hypothesis to discovery. *arXiv preprint arXiv:2505.16477*, 2025b.

Yilan Zhang, Yingxue Xu, Jianqi Chen, Fengying Xie, and Hao Chen. Prototypical information bottlenecking and disentangling for multimodal cancer survival prediction. *arXiv preprint arXiv:2401.01646*, 2024d.

Mingyang Zhao, Gaofeng Meng, and Dong-Ming Yan. Occlusion-aware non-rigid point cloud registration via unsupervised neural deformation correntropy. In *The Thirteenth International Conference on Learning Representations*, 2025.

Kaixiang Zheng and En-Hui Yang. Knowledge distillation based on transformed teacher matching. *arXiv preprint arXiv:2402.11148*, 2024.

Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. A survey of large language models for code: Evolution, benchmarking, and future trends. *ArXiv*, abs/2311.10372, 2023. URL `https://api.semanticscholar.org/CorpusId:265281389`.

Allan Zhou, Chelsea Finn, and James Harrison. Universal neural functionals. *Advances in neural information processing systems*, 37:104754–104775, 2024.

Junjie Zhou, Jiao Tang, Yingli Zuo, Peng Wan, Daoqiang Zhang, and Wei Shao. Robust multimodal survival prediction with the latent differentiation conditional variational autoencoder. *arXiv preprint arXiv:2503.09496*, 2025.

Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. Codebertscore: Evaluating code generation with pretrained models of code. *arXiv preprint arXiv:2302.05527*, 2023.

Chaoyi Zhu, Jiayi Tang, Jeroen M. Galjaard, Pin-Yu Chen, Robert Birke, Cornelis Bos, and Lydia Y. Chen. Tabwak: A watermark for tabular diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL `https://openreview.net/forum?id=71pur4y8gs`.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 14901–14911, 2025b.

Minjun Zhu, Qiujie Xie, Yixuan Weng, Jian Wu, Zhen Lin, Linyi Yang, and Yue Zhang. Ai scientists fail without strong implementation capability. *arXiv preprint arXiv:2506.01372*, 2025c.

Xiangyang Zhu, Renrui Zhang, Bowei He, Ziyu Guo, Ziyao Zeng, Zipeng Qin, Shanghang Zhang, and Peng Gao. Pointclip v2: Prompting clip and gpt for powerful 3d open-world learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2639–2650, 2023.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. Embedllm: Learning compact representations of large language models. *arXiv preprint arXiv:2410.02223*, 2024.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

Henry Peng Zou, Wei-Chieh Huang, Yaozu Wu, Yankai Chen, Chunyu Miao, Hoang Nguyen, Yue Zhou, Weizhi Zhang, Liancheng Fang, Langzhou He, et al. Llm-based human-agent collaboration and interaction systems: A survey. *arXiv preprint arXiv:2505.00753*, 2025a.

Henry Peng Zou, Wei-Chieh Huang, Yaozu Wu, Yankai Chen, Chunyu Miao, Hoang Nguyen, Yue Zhou, Weizhi Zhang, Liancheng Fang, Langzhou He, et al. A survey on large language model based human-agent systems. *arXiv preprint arXiv:2505.00753*, 2025b.

Henry Peng Zou, Wei-Chieh Huang, Yaozu Wu, Chunyu Miao, Dongyuan Li, Aiwei Liu, Yue Zhou, Yankai Chen, Weizhi Zhang, Yangning Li, et al. A call for collaborative intelligence: Why human-agent systems should precede ai autonomy. *arXiv preprint arXiv:2506.09420*, 2025c.

A LLM Usage Statement

In this work, LLMs were used only to aid or polish writing. Specifically, initial drafts of the manuscript were written by the authors, after which LLMs were employed to enhance clarity and richness of expression. All content was subsequently reviewed and revised by the authors, and the final version reflects the authors' own corrections and approval. No parts of the research design, experiments, analysis, or results relied on LLMs.

B Benchmark Statistics

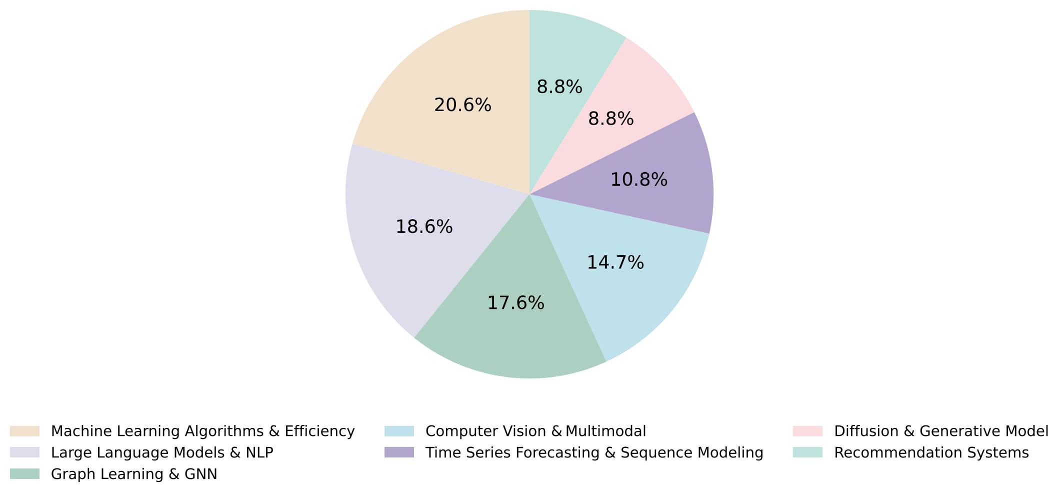


Figure 3: The domain of the tasks within RECODE-H.

Figure 3 illustrates the distribution of task domains in RECODE. The benchmark covers a diverse range of research areas. This balanced coverage highlights RECODE-H emphasis on capturing the breadth of modern AI research.

C Data Contamination Analysis

A common concern in benchmarks is that the evaluated LLMs may have been exposed to the source papers or repositories during pre-training, particularly for works published before late 2023. To examine whether such contamination materially affects model performance, we analyze results stratified by the publication year of the underlying research papers. Our dataset includes tasks from 2023, 2024, and 2025, enabling a coarse but meaningful assessment of year dependent performance trends.

Table 4 reports evaluation metrics across years for seven LLMs. If pre-training contamination were a dominant factor, models would be expected to show substantially higher performance on 2023 tasks, which are the most likely to appear in pre-training corpora. Instead, we observe the opposite pattern across nearly all models: performance on 2023 tasks is consistently the lowest, 2024 tasks improve markedly, and 2025 tasks often match or exceed 2024 levels—despite being too recent to be included in any model's training data.

These findings suggest that contamination is unlikely to be a primary driver of model performance. Instead, differences across years more plausibly reflect (1) evolving research trends, (2) variation in methodological complexity, and (3) stronger alignment between newer tasks and modern model capabilities. While eliminating contamination entirely is infeasible for frontier models trained on broad web corpora, this analysis indicates that leakage does not explain the performance patterns observed in RECODE-H.

Model	Year	MRR	Recall	Test Case	CodeBLEU	CodeBERT
GPT-5	2023	0.116	0.588	0.806	0.297	0.884
	2024	0.230	0.685	0.847	0.322	0.890
	2025	0.277	0.833	0.853	0.318	0.888
GPT-5-mini	2023	0.065	0.294	0.659	0.288	0.890
	2024	0.187	0.741	0.856	0.306	0.890
	2025	0.216	0.733	0.806	0.329	0.889
GPT-5-nano	2023	0.051	0.176	0.490	0.275	0.885
	2024	0.097	0.352	0.562	0.301	0.853
	2025	0.106	0.433	0.536	0.283	0.849
DeepSeek	2023	0.111	0.588	0.802	0.342	0.901
	2024	0.212	0.704	0.817	0.342	0.897
	2025	0.234	0.767	0.830	0.343	0.896
Gemini-2.5 Pro	2023	0.127	0.588	0.723	0.332	0.897
	2024	0.202	0.574	0.676	0.336	0.891
	2025	0.154	0.600	0.740	0.330	0.879
Gemini-2.5 Flash	2023	0.090	0.471	0.727	0.331	0.903
	2024	0.161	0.630	0.782	0.355	0.900
	2025	0.128	0.567	0.784	0.354	0.896
Claude	2023	0.087	0.412	0.561	0.336	0.891
	2024	0.109	0.500	0.675	0.334	0.894
	2025	0.130	0.467	0.622	0.337	0.892

Table 4: Performance of LLMs across paper publication years.

D ReCodeAgent Tool Design

Recent LLM agent research has rapidly expanded to include advanced agent strategies (Yao et al., 2023; Huang & Caragea, 2025; Zou et al., 2025a;c), memory mechanisms(Huang et al., 2026), enhanced reasoning paradigms(Zhang et al., 2025a; Yang et al., 2025), and safety controls(Huang et al., 2025). While these directions improve capability, they also introduce architectural complexity and evaluation confounds. In contrast, ReCodeAgent follows a minimal design principle, restricting the agent to a small, well-defined set of repository interaction tools. This lightweight design reduces hidden state and emergent behaviors, improves reproducibility, and isolates the core problem of iterative code refinement under structured tool constraints. ReCodeAgent interacts with the repository through a fixed set of tools that enable file inspection, code retrieval, rewriting, and execution based validation. These tools define the agent's action space and support reproducible multi-turn code refinement.

READ. The `READ` tool returns the full contents of any repository file, allowing the agent to recover context and understand existing implementations before making changes.

RETRIEVE. The `RETRIEVE` tool retrieves the most relevant functions or code regions given a natural-language query, enabling the agent to locate where specific functionality is implemented across the repository.

REPLACE. The `REPLACE` tool rewrites the entire contents of the target file with newly generated code. Although earlier versions of the system also included a span-based `EDIT` tool for detailed modifications, empirical evaluation showed that models frequently misidentified edit boundaries or produced inconsistent patches, resulting in structural corruption. Whole-file rewriting via `REPLACE` consistently yielded more reliable results. Therefore, `REPLACE` is retained as the sole editing mechanism.

SUBMIT. The `SUBMIT` tool executes the current implementation and returns unit test outcomes and execution logs, providing the diagnostic signals that guide iterative refinement.

BROWSE. The `BROWSE` tool lists all files and directories in the repository, allowing the agent to inspect project structure and locate relevant modules.
These tools support ReCodeAgent's iterative workflow: inspecting repository context, locating rel-

evant code, applying revisions, and validating correctness. This structured tool interface ensures transparency and reproducibility in evaluating research code development.

E EVALUATION OF GENERATED FEEDBACK

We additionally conduct a small-scale study using real human feedback. We collect 50 intermediate human and LLM interaction samples in which annotators provide natural language feedback on code that fails unit tests, following the same iterative protocol used for LLM augmented feedback. Figure 4 shows that human feedback yields measurable but consistently smaller performance gains compared with LLM-augmented expert feedback. Human annotations are typically partial, noisy and identifying only surface level issues and lacking the deeper diagnostic reasoning or prescriptive correction that LLM augmented feedback provides. This qualitative gap is consistent with prior observations from MINT(Wang et al., 2023b) and CONVCODEWORLD(Han et al., 2025).
Although our benchmark uses simulated expert feedback for reproducibility, we acknowledge that it does not capture the full variability of real human feedback, including imprecision, underspecification, and heterogeneity in expertise or communication style. Our framework can in principle incorporate these phenomena by introducing controlled perturbations—such as probabilistic error injection, incomplete messages, stylistic variation, or expertise-conditioned feedback distributions—within the existing feedback-sampling pipeline.
We restrict human annotation to this analysis due to the high cost and variability of real feedback, which limits scalable and reproducible evaluation. LLM-augmented feedback therefore serves as a standardized and repeatable reference for the main benchmark, while extending the framework with more realistic noise remains an important direction for future work.

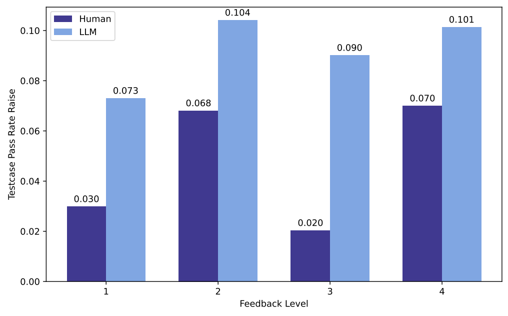


Figure 4: The rise of average pass rate of GPT−5 when guided by GPT-o4-mini and real human feedback.

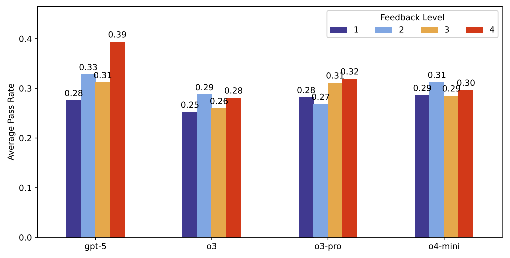


Figure 5: Average pass rate of GPT−5−mini when guided by different feedback models across feedback levels. GPT−5 feedback yields the strongest improvements, particularly at Level 4.

F ABLATION STUDY ON FEEDBACK MODEL

F.1 FEEDBACK QUALITY

We conduct an ablation study to evaluate how different reasoning models used for feedback generation influence the overall interactive code generation process. Specifically, we test GPT-5-mini as the code generator while varying the feedback model among GPT−5, GPT−o3, GPT−o3−pro, and GPT−o4−mini.

Table 5 reports the average test case pass rates across these settings. Overall,GPT−5−mini achieves pass rates around 30% when guided by external feedback. Among the feedback models, GPT-5 provides the strongest improvements, yielding the highest pass rate 32%. GPT−o3−pro and GPT−o4−mini produce comparable results around 30%, while GPT−o3 feedback leads to the lowest performance of 27%.

Figure 5 further examines how pass rates evolve across different feedback guidance levels. The trends for GPT−5, GPT−o3, and GPT−o4−mini align with those observed in Table 2. Notably, GPT-5 demonstrates a clear advantage at Level 4 feedback, achieving a significant improvement over

Feedback Model	Pass Rate
GPT-5	0.32
GPT-o3	0.27
GPT-o3-pro	0.30
GPT-o4-mini	0.30

Table 5: Ablation study of average pass rate using different feedback models.

other models. This highlights `GPT-5`'s stronger ability to provide precise and corrective code-level feedback that accelerates convergence to correct implementations.

F.2 FEEDBACK COST

Model	Avg. Input Tokens	Avg. Output Tokens	Avg. API Cost($)
GPT-5	20 098.62	7781.91	0.102
GPT-o3	20 099.38	2997.10	0.064
GPT-o3-pro	20 089.31	3267.83	0.663
GPT-o4-mini	20 103.95	5060.34	0.044

Table 6: Comparison of average token usage and API cost across different feedback models.

In this section we compare different feedback model in the aspect of cost. Table 6 shows that when evaluated on the same tasks, all feedback models process a similar number of input tokens, but differ in output size and API cost. GPT-5 provides the most effective feedback but at a moderate cost (0.102 $ per feedback), while o3-pro is disproportionately expensive (0.663 $). In contrast, o3 and especially o4-mini deliver much lower costs, with o4-mini offering the best cost-efficiency.

G DATA LEAKAGE ANALYSIS

In generating feedback, we provided the LLMs with canonical code annotated with explicit comments to ensure correctness. While necessary for accurate feedback, this setup raises concerns about potential code leakage, particularly at feedback Levels 1-3. At these levels, the LLMs are expected to offer natural language guidance for correcting errors rather than reproducing ground truth implementations. To assess the risk, we sampled 1,000 feedback instances and evaluated leakage by detecting occurrences of ground truth code snippets within the feedback.

Our analysis shows that code leakage at Levels 1-3 is negligible, remaining below 2%. Results across models are summarized in Table 7.

Model	Level 1	Level 2	Level 3	Level 4
GPT-5	0	0	0.00	0.020
GPT-o3	0	0	0.01	0.305
GPT-o4-mini	0	0	0.00	0.195
GPT-o3-pro	0	0	0.02	0.330

Table 7: Code leakage rates across models. Leakage remains negligible ($< 1\%$) for Levels 1-3, but is more pronounced at Level 4 where ground-truth snippets are explicitly provided.

We further analyzed the extent of coverage—that is, the proportion of canonical code revealed in leaked snippets. These results are reported in Table 8. Again, leakage is effectively absent at Levels 1-2, minimal at Level 3, and substantially higher at Level 4.

H EXTENDED ANALYSIS UNDER INCREASED INTERACTION BUDGETS

To further characterize model performance under varying feedback conditions, we extend the maximum number of researcher LLM interaction rounds from 10 to 15. This extended setting allows us to examine whether remaining errors stem from insufficient feedback or from iterative integration dynamics.

Model	Level 1	Level 2	Level 3	Level 4
GPT-5	0	0	0.00	0.245
GPT-o3	0	0	0.02	0.837
GPT-o4-mini	0	0	0.00	0.478
GPT-o3-pro	0	0	0.01	0.676

Table 8: Proportion of canonical code covered by leaked snippets. Leakage coverage is near-zero for Levels 1-3, but increases significantly at Level 4.

Table 9 reports Recall@15 for GPT-5 across all feedback levels. Performance consistently improves with additional turns, indicating that multi-turn refinement continues to contribute meaningfully beyond the default horizon.

Feedback Level	**Recall@15 (GPT-5)**
Level 0	0.549
Level 1	0.779
Level 2	0.880
Level 3	0.963
Level 4	**0.972**

Table 9: Recall@15 for GPT-5 under different feedback levels.

Lower feedback levels exhibit the largest marginal gains. *Level 0*, which provides only execution logs, improves substantially but remains constrained by weak error localization. *Level 1* and *Level 2* benefit from the extended horizon as well, reflecting that coarse and mid-level diagnostic information often requires multiple refinement cycles before errors are resolved.

Higher feedback levels demonstrate accelerated convergence. *Level 3* approaches saturation by 15 turns, driven by the effectiveness of targeted natural-language correction. *Level 4* reaches near-complete correctness, with remaining discrepancies attributable to multi-step repository integration, such as interface alignment and dependency resolution, rather than missing supervisory information.

The extended turn evaluation indicates that richer feedback enables faster and more reliable convergence, while lower levels remain limited by error localization and conceptual understanding challenges.

I Error type analysis

The Table 3 reveals that while all models share common categories of mistakes, the distribution of these errors varies significantly across LLM families, producing distinct patterns of weakness. Broadly, low-level syntax and runtime errors are relatively uncommon, showing that most modern models have surpassed the stage of struggling with surface level coding rules. Instead, the majority of failures stem from higher level challenges, such as faithfully interpreting research descriptions or bridging implicit domain knowledge.

Within the GPT family, model size plays a decisive role. The largest variants, `GPT-5` and `GPT-5-mini`, produce relatively few syntax mistakes, with their errors dominated by missing knowledge and instruction misalignment. This shows that scaling tends to reduce shallow failures while pushing the challenge into semantic fidelity. By contrast, `GPT-5-nano` displays a very different profile: it is far more prone to low-level bugs and misunderstandings, indicating that smaller models lack the stability and consistency to translate complex descriptions into working code.

The Gemini models demonstrate another interesting split. `Gemini-2.5-flash` resembles larger GPT models, with its main failures concentrated in missing knowledge and context, reflecting a tendency to overlook implicit assumptions or repository-specific conventions. In contrast, `Gemini-2.5-pro` is far more vulnerable to instruction misunderstandings, producing errors that stem from misreading or misapplying the core methodological steps. This divergence between two variants of the same family highlights that architectural or training differences can lead to fundamentally distinct error patterns.

DeepSeek and Claude each illustrate contrasting limitations. `DeepSeek-V3.1` shows relatively balanced distributions of misunderstandings and knowledge gaps. `Claude-Sonnet-4`, on the other hand, stands out for having the highest rate of syntax and runtime errors among all models. While it shares semantic weaknesses with other LLMs, its disproportionate low-level fragility undermines reliability and signals weaker baseline robustness in code execution.

Taken together, these patterns emphasize that while the frontier of error types has shifted away from syntax toward semantic fidelity, different model families exhibit characteristic signatures of failure. Some models, like `GPT-5`, excel at suppressing shallow mistakes yet still falter on implicit domain knowledge, while others, like `Gemini-2.5-pro` or `Claude-Sonnet-4`, expose deeper struggles with faithfully grounding research instructions or ensuring execution stability.

J FEEDBACK ADOPTION ANALYSIS TABLE

Table10 provides statistics on feedback adoption and error resolve reate across different feedback levels. Table11 provides statistics on feedback adoption and error resolve reate across different error types.

Model	**Feedback Level**	**A** (%)	**NA** (%)	**AS** (%)	**AP** (%)	**ANS** (%)	**NAS** (%)	**NASP** (%)	**NANS** (%)
GPT-5	1	80.2	19.8	50.9	5	24.3	0.5	0.5	18.9
GPT-5	2	93	7	65.4	3.7	23.8	0	0	7
GPT-5	3	91	9	66.3	6	18.6	0.5	0	8.5
GPT-5	4	91.8	8.2	65	5.5	21.4	0	0	8.2
GPT-5-mini	1	68.8	31.2	35.9	3.5	29.4	0.9	0	30.3
GPT-5-mini	2	87.1	12.9	55.4	5	26.7	0	0	12.9
GPT-5-mini	3	96.3	3.7	61.9	10.2	24.2	0	0	3.7
GPT-5-mini	4	97.6	2.4	73.8	4.8	19	0	0	2.4
GPT-5-nano	1	45.2	54.8	30	4.6	10.6	0	0	54.8
GPT-5-nano	2	68.1	31.9	38.9	7.4	21.8	0	0	31.9
GPT-5-nano	3	68.7	31.3	45.7	5.3	17.7	1.2	0	30
GPT-5-nano	4	75.6	24.4	45.9	7	22.7	0	0	24.4
Gemini-2.5-pro	1	51.5	48.5	36.1	3	12.4	0	0	48.5
Gemini-2.5-pro	2	65.8	34.2	46.5	3.9	15.4	0	0	34.2
Gemini-2.5-pro	3	58.5	41.5	35.1	8.3	15.1	0	0	41.5
Gemini-2.5-pro	4	58.9	41.1	51.3	1.5	6.1	0.5	0	40.6
Gemini-2.5-flash	1	80.7	19.3	40.6	7.5	32.5	0	0	19.3
Gemini-2.5-flash	2	88.4	11.6	46.9	6.3	35.3	0	0	11.6
Gemini-2.5-flash	3	94.1	5.9	65.1	4.3	24.7	0.5	0	5.4
Gemini-2.5-flash	4	95.6	4.4	62.1	5.5	28	0	0	4.4
DeepSeek-V3.1	1	75.3	24.7	47.1	7.3	20.8	0	0	24.7
DeepSeek-V3.1	2	87.9	12.1	51.8	7.3	28.7	0	0	12.1
DeepSeek-V3.1	3	88.1	11.9	65.2	6.3	16.7	0	0	11.9
DeepSeek-V3.1	4	86.8	13.2	68.8	5.1	12.9	0	0	13.2
Claude-sonnet-4	1	82	18	64.6	7.8	9.7	0.5	0.5	17
Claude-sonnet-4	2	71.4	28.6	57.6	3.3	10.5	0.5	0	28.1
Claude-sonnet-4	3	67.8	32.2	55.1	3.4	9.3	0.5	0.5	31.2
Claude-sonnet-4	4	77.6	22.4	68.3	2	7.3	0.5	0	22

Table 10: Adoption and error-resolution outcomes across models and guidance levels. Columns denote: A = adopted (%), NA = non-adopted (%), AS = adopted and solved (%), AP = adopted and partially solved (%), ANS = adopted and not solved (%), NAS = non-adopted and solved (%), NASP = non-adopted and partially solved (%), and NANS = non-adopted and not solved (%).

Model	**Error Type**	**A** (%)	**NA** (%)	**AS** (%)	**AP** (%)	**ANS** (%)	**NAS** (%)	**NASP** (%)	**NANS** (%)
GPT-5	T1	96.7	3.3	72.5	2.2	22	0	0	3.3
GPT-5	T2	87.9	12.1	52.4	6.6	28.9	0.4	0.4	11.4
GPT-5	T3	89.1	10.9	66.3	4.5	18.4	0.2	0	10.7
GPT-5	T4	85.7	14.3	68.6	5.7	11.4	0	0	14.3
GPT-5-mini	T1	94.7	5.3	66.3	3.2	25.3	0	0	5.3
GPT-5-mini	T2	78.5	21.5	50.2	5.8	22.4	0.9	0	20.6
GPT-5-mini	T3	89	11	56.7	6.2	26.2	0	0	11
GPT-5-mini	T4	96.1	3.9	51	9.8	35.3	0	0	3.9
GPT-5-nano	T1	69.9	30.1	53.9	6.2	9.8	0.5	0	29.5
GPT-5-nano	T2	56.1	43.9	30.3	6.4	19.4	0.3	0	43.6
GPT-5-nano	T3	66.7	33.3	40.1	6.2	20.4	0	0	33.3
GPT-5-nano	T4	65.6	34.4	45.3	1.6	18.8	0	0	34.4
Gemini-2.5-pro	T1	72.7	27.3	54.7	3.9	14.1	0	0	27.3
Gemini-2.5-pro	T2	44.7	55.3	32.7	3.5	8.5	0	0	55.3
Gemini-2.5-pro	T3	63.7	36.2	44.7	4.4	14.7	0.3	0	35.9
Gemini-2.5-pro	T4	77.6	22.4	52.2	7.5	17.9	0	0	22.4
Gemini-2.5-flash	T1	93.5	6.5	69.1	2.4	22	0.8	0	5.7
Gemini-2.5-flash	T2	90.5	9.5	51.8	10.6	28.1	0	0	9.5
Gemini-2.5-flash	T3	88.3	11.7	48.4	4.3	35.6	0	0	11.7
Gemini-2.5-flash	T4	87.3	12.7	54	6.3	27	0	0	12.7
DeepSeek-V3.1	T1	87.4	12.6	70.5	2.4	14.5	0	0	12.6
DeepSeek-V3.1	T2	83.3	16.7	54.7	5	23.6	0	0	16.7
DeepSeek-V3.1	T3	83.5	16.5	56.3	7.9	19.3	0	0	16.5
DeepSeek-V3.1	T4	85.3	14.7	48	14.7	22.7	0	0	14.7
Claude-sonnet-4	T1	79.4	20.6	62.6	3.3	13.6	0.5	0	20.1
Claude-sonnet-4	T2	68.2	31.8	55.9	3.1	9.2	0	0	31.8
Claude-sonnet-4	T3	78.8	21.2	67.8	6.2	4.8	1.1	0.4	19.8
Claude-sonnet-4	T4	67.2	32.8	52.5	3.3	11.5	0	1.6	31.1

Table 11: Adoption and error-resolution outcomes across models and error type/ Columns denote: A = adopted (%), NA = non-adopted (%), AS = adopted and solved (%), AP = adopted and partially solved (%), ANS = adopted and not solved (%), NAS = non-adopted and solved (%), NASP = non-adopted and partially solved (%), and NANS = non-adopted and not solved (%).

K Paper List

The paper list are in Tables 12, 13, 14 and 15.

Paper Title	Venue
1. PEDANTS: Cheap but Effective and Interpretable Answer Equivalence (Li et al., 2024f)	EMNLP 2024
2. Differential Transformer (Ye et al., 2024)	ICLR 2025
3. Mapping the Multiverse of Latent Representations (Wayland et al., 2024)	ICML 2024
4. TABDIFF: A Mixed-Type Diffusion Model for Tabular Data Generation (Shi et al., 2024)	ICLR 2025
5. Mapping the Multiverse of Latent Representations (Wayland et al., 2024)	ICML 2024
6. Task-Agnostic Machine-Learning-Assisted Inference (Miao & Lu, 2024)	NIPS 2024
7. Universal Neural Functionals (Zhou et al., 2024)	NeurIPS 2024
8. Dendritic Integration Inspired Artificial Neural Networks Capture Data Correlation (Liu et al., 2023a)	NeurIPS 2024
9. Transformers without Normalization (Zhu et al., 2025b)	CVPR 2025
10. SimPO: Simple Preference Optimization with a Reference-Free Reward (Meng et al., 2024)	NeurIPS 2024
11. Knowledge Distillation Based on Transformed Teacher Matching (Zheng & Yang, 2024)	ICLR 2024
12. Data Unlearning in Diffusion Models (Alberti et al., 2025)	ICLR 2025
13. It's Never Too Late: Fusing Acoustic Information into Large Language Models for Automatic Speech Recognition (Chen et al., 2024a)	ICLR 2024
14. GraphMETRO: Mitigating Complex Graph Distribution Shifts via Mixture of Aligned Experts (Wu et al., 2023)	NeurIPS 2024
15. Balanced Data Sampling for Language Model Training with Clustering (Shao et al., 2024)	ACL 2024
16. LESS: Selecting Influential Data for Targeted Instruction Tuning (Xia et al., 2024)	ICML 2024
17. Copula Conformal Prediction for Multi-step Time Series Forecasting (Sun & Yu, 2022)	ICLR 2024
18. Conformal Language Model Reasoning with Coherent Factuality (Rubin-Toles et al., 2025)	ICLR 2025
19. Data-efficient Fine-tuning for LLM-based Recommendation (Lin et al., 2024a)	SIGIR 2024
20. Language Models with Conformal Factuality Guarantees (Mohri & Hashimoto, 2024)	ICML2024
21. Universal Neural Functionals (Zhou et al., 2024)	NIPS 2024
22. Frequency-domain MLPs are More Effective Learners in Time Series Forecasting (Yi et al., 2023)	NIPS 2023
23. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers (Nie, 2022)	ICLR 2023
24. Are Transformers Effective for Time Series Forecasting? (Zeng et al., 2023)	AAAI 2023
25. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting (Liu et al., 2023c)	ICLR 2024
26. TimeMixer: Decomposable Multiscale Mixing for Time Series Forecasting (Wang et al., 2024a)	ICLR 2024

Table 12: Paper list part (a)

Paper Title	Venue
27. TimeXer: Empowering Transformers for Time Series Forecasting with Exogenous Variables (Wang et al., 2024a)	NeurIPS 2024
28. Large Language Models Are Zero-Shot Time Series Forecasters (Gruver et al., 2023)	NIPS 2023
29. ARC: A Generalist Graph Anomaly Detector with In-Context Learning (Liu et al., 2024d)	NIPS 2024
30. An Efficient Matrix Multiplication Algorithm for Accelerating Inference in Binary and Ternary Neural Networks (Dehghankar et al., 2024)	ICML 2025
31. FairHash: A Fair and Memory/Time-efficient Hashmap (Asudeh et al., 2023)	VLDB 2024
32. MALT Powers Up Adversarial Attacks (Melamed et al., 2024)	NeurIPS 2024
33. End-to-end Learnable Clustering for Intent Learning in Recommendation (Liu et al., 2024e)	NeurIPS 2024
34. ContextGNN: Beyond Two-Tower Recommendation Systems (Yuan et al., 2024)	ICLR 2025
35. Wukong: Towards a Scaling Law for Large-Scale Recommendation (Zhang et al., 2024a)	ICML 2024
36. Preference Diffusion for Recommendation (Liu et al., 2024c)	ICLR 2025
37. Augmenting Sequential Recommendation with Balanced Relevance and Diversity (Dang et al., 2025)	AAAI 2025
38. Why Does Dropping Edges Usually Outperform Adding Edges in Graph Contrastive Learning? (Xu et al., 2025)	AAAI 2025
39. Simple and Asymmetric Graph Contrastive Learning without Augmentations (Xiao et al., 2023)	NeurIPS 2023
40. Hierarchical Topology Isomorphism Expertise Embedded Graph Contrastive Learning (Li et al., 2024c)	AAAI 2024
41. Simple and Asymmetric Graph Contrastive Learning without Augmentations (Xiao et al., 2023)	NeurIPS 2023
42. HomoGCL: Rethinking Homophily in Graph Contrastive Learning (Li et al., 2023)	KDD 2023
43. SmoothGNN: Smoothing-aware GNN for Unsupervised Node Anomaly Detection (Dong et al., 2024)	WWW 2025
44. Representation Alignment for Generation: Training Diffusion Transformers Is Easier Than You Think (Yu et al., 2024)	ICLR 2025
45. LoRAMoE: Revolutionizing Mixture of Experts for Maintaining World Knowledge in Language Model Alignment (Dou et al., 2023)	ACL 2024
46. Reconstruction vs. Generation: Taming Optimization Dilemma in Latent Diffusion Models (Yao et al., 2025)	CVPR 2025
47. Patch-wise Structural Loss for Time Series Forecasting! (Kudrat et al., 2025)	ICML 2025
48. Parameter-Efficient Fine-Tuning with Discrete Fourier Transform (Gao et al., 2024)	ICML 2024
49. Attractor Memory for Long-Term Time Series Forecasting: A Chaos Perspective (Hu et al., 2024)	NeurIPS 2024
50. FITS: Modeling Time Series with 10k Parameters (Xu et al., 2023)	ICLR 2024
51. Learning Versatile Skills with Curriculum Masking (Tang et al., 2024)	NeurIPS 2024
52. Federated Transformer: Multi-Party Vertical Federated Learning on Practical Fuzzily Linked Data (Wu et al., 2024b)	NeurIPS 2024

Table 13: Paper list part (b)

Paper Title	Venue
53. In-context Time Series Predictor (Lu et al., 2024)	ICLR 2025
54. Retaining Key Information under High Compression Ratios: Query-Guided Compressor for LLMs (Cao et al., 2024)	ACL 2024
55. Estimating Conditional Mutual Information for Dynamic Feature Selection (Gadgil et al., 2023)	ICLR 2024
56. Deep Temporal Graph Clustering (Liu et al., 2023b)	ICLR 2024
57. DigiRL: Training In-The-Wild Device-Control Agents with Autonomous Reinforcement Learning (Bai et al., 2024)	NeurIPS 2024
58. Unsupervised Multimodal Clustering for Semantics Discovery in Multimodal Utterances (Zhang et al., 2024b)	ACL 2024
59. Occlusion-aware Non-Rigid Point Cloud Registration via Unsupervised Neural Deformation Correntropy (Zhao et al., 2025)	ICLR 2025
60. Mixed-Type Tabular Data Synthesis with Score-based Diffusion in Latent Space (Zhang et al., 2023a)	ICLR 2024
61. ReMasker: Imputing Tabular Data with Masked Autoencoding (Du et al., 2023)	ICLR 2024
62. TabWak: A Watermark for Tabular Diffusion Models (Zhu et al., 2025a)	ICLR 2025
63. TAROT: Targeted Data Selection via Optimal Transport (Feng et al., 2024)	CVPR 2024
64. Learning Divergence Fields for Shift-Robust Graph Representations (Wu et al., 2024a)	ICML 2024
65. Supercharging Graph Transformers with Advective Diffusion Wu et al. (2025a)	ICML 2025
66. DiffPuter: Empowering Diffusion Models for Missing Data Imputation (Zhang et al., 2024c)	ICLR 2025
67. Fair Set Cover (Dehghankar et al., 2025)	KDD 2025
68. The Road Less Scheduled (Defazio et al., 2024)	Neurips 2024
69. SLAB: Efficient Transformers with Simplified Linear Attention and Progressive Re-parameterized Batch Normalization (Guo et al., 2024)	ICML 2024
70. Turning up the Heat: MIN-p Sampling for Creative and Coherent LLM Outputs (Nguyen et al., 2024)	ICLR 2025
71. Data Unlearning in Diffusion Models (Alberti et al., 2025)	ICLR 2025
72. Your ViT is Secretly an Image Segmentation Model (Kerssies et al., 2025)	CVPR 2025
73. TABDIFF: A Mixed-Type Diffusion Model for Tabular Data Generation (Shi et al., 2024)	ICLR 2025
74. Bipartite Graph Convolutional Hashing for Effective and Efficient Top-N Search in Hamming Space (Chen et al., 2023b)	WWW 2023
75. Semi-supervised Node Importance Estimation with Informative Distribution Modeling for Uncertainty Regularization (Chen et al., 2025a)	WWW 2025
76. Towards Effective Top-N Hamming Search via Bipartite Graph Contrastive Hashing (Chen et al., 2024b)	TKDE 2024
77. Hyperbolic Image-Text Representations (Desai et al., 2023)	ICML 2023
78. Modeling Dense Multimodal Interactions Between Biological Pathways and Histology for Survival Prediction (Jaume et al., 2024)	CVPR 2024

Table 14: Paper list part (c)

Paper Title	Venue
79. Multimodal optimal transport-based co-attention transformer with global structure consistency for survival prediction (Xu & Chen, 2023)	ICCV 2023
80. MoME: Mixture of Multimodal Experts for Cancer Survival Prediction (Xiong et al., 2024)	MICCAI 2024
81. Prototypical Information Bottlenecking and Disentangling for Multimodal Cancer Survival Prediction (Zhang et al., 2024d)	ICLR 2024
82. Interpretable Vision-Language Survival Analysis with Ordinal Inductive Bias for Computational Pathology (Liu et al., 2024a)	ICLR 2025
83. Multimodal Prototyping for Cancer Survival Prediction (Song et al., 2024)	ICML 2024
84. Robust Multimodal Survival Prediction with the Latent Differentiation Conditional Variational AutoEncoder (Zhou et al., 2025)	CVPR 2025
85. InfoBatch: Lossless Training Speed Up by Unbiased Dynamic Data Pruning (Qin et al., 2023)	ICLR 2024
86. UniGAD: Unifying Multi-level Graph Anomaly Detection (Lin et al., 2024b)	NeurIPS 2024
87. PaGE-Link: Path-based Graph Neural Network Explanation for Heterogeneous Link Prediction (Zhang et al., 2023b)	WWW 2023
88. Rethinking Fair Graph Neural Networks from Re-balancing (Li et al., 2024e)	KDD 2024
89. LightGCL: Simple Yet Effective Graph Contrastive Learning for Recommendation (Cai et al., 2023)	ICLR 2023
90. Enhancing Activity Prediction Models in Drug Discovery with the Ability to Understand Human Language (Seidl et al., 2023)	ICML 2023
91. What to align in multimodal contrastive learning? (Dufumier et al., 2024)	ICLR 2025
92. AlphaFuse: Learn ID Embeddings for Sequential Recommendation in Null Space of Language Embeddings (Hu et al., 2025)	SIGIR 2025
93. Beyond Matryoshka: Revisiting Sparse Coding for Adaptive Representation (Wen et al., 2025)	ICML 2025
94. EmbedLLM: Learning Compact Representations of Large Language Models (Zhuang et al., 2024)	ICLR 2025
95. Highly Compressed Tokenizer Can Generate Without Training (Beyer et al., 2025)	ICML 2025
96. LLM-ESR: Large Language Models Enhancement for Long-tailed Sequential Recommendation (Liu et al., 2024b)	NeurIPS 2024
97. Offline Multi-Agent Reinforcement Learning with Implicit Global-to-Local Value Regularization (Wang et al., 2023a)	NeurIPS 2023
98. Enhancing Large Language Models in Coding Through Multi-Perspective Self-Consistency (Huang et al., 2024)	ACL 2024
99. Context Matters: Enhancing Sequential Recommendation with Context-aware Diffusion-based Contrastive Learning (Cui et al., 2024b)	CIKM 2024
100. Treatment-Aware Hyperbolic Representation Learning for Causal Effect Estimation with Social Networks (Cui et al., 2024a)	SDM 2024
101. PointCLIP V2: Prompting CLIP and GPT for Powerful 3D Open-world Learning (Zhu et al., 2023)	ICCV 2023
102. LLM-Pruner: On the Structural Pruning of Large Language Models (Ma et al., 2023)	NeurIPS 2023

Table 15: Paper list part (d)

L CASE STUDY

In this section, we provide a step-by-step demonstration of how feedback progressively guides model-generated code toward the correct canonical implementation. We begin with the formal paper description (Fig. 6), which specifies the underlying algorithm and serves as the authoritative reference. From this description, a detailed instruction is constructed, translating the theoretical requirements into a precise programming task. The model's initial output, shown as generated code (Fig. 8), captures the general intent but diverges from the canonical code (Fig. 7) in subtle yet important ways, such as in the handling of `min_tokens_to_keep`. To close this gap, the feedback (Fig. 9) is produced, which diagnoses the deviation, explains its consequences, and prescribes a targeted correction using a scatter-based enforcement strategy. Finally, we present the revised code (Fig. 10), where the integration of feedback yields an implementation that fully aligns with the canonical method.

Paper Description

```
Formally, at each time step $t$, let $\mathcal{V}$ denote the vocabulary, and $P(x_t \mid x_
    {1:t-1})$ represent the conditional probability distribution over the vocabulary for the
    next token $x_t$. Min-\( p \) sampling works as follows:
\begin{enumerate}
    \item \textbf{Calculate the Maximum Probability:} Identify the maximum probability token
        in the distribution, denoted as $p_{\max} = \max_{v \in \mathcal{V}} P(v \mid x_{1:t
        -1})$.
    \item \textbf{Define the Truncation Threshold:} Set a base probability threshold, $p_{\
        text{base}} \in (0, 1]$, and scale it by $p_{\max}$ to determine the actual
        truncation threshold:
    \begin{equation}
        p_{\text{scaled}} = p_{\text{base}} \times p_{\max}
    \end{equation}
    This threshold ensures that tokens with sufficiently high relative probabilities are
        considered while filtering out less probable tokens in a context-dependent manner.
    \item \textbf{Define the Sampling Pool:} Construct the sampling pool $\mathcal{V}_{\text{
        min}}$ consisting of tokens whose probabilities are greater than or equal to $p_{\
        text{scaled}}$:
    \begin{equation}
        \mathcal{V}_{\text{min}} = \{v \in \mathcal{V} : P(v \mid x_{1:t-1}) \geq p_{\text{
            scaled}} \}
    \end{equation}
    \item \textbf{Sample from the Pool:} Sample the next token $x_t$ from the reduced set $\
        mathcal{V}_{\text{min}}$ according to their normalized probabilities:
    \begin{equation}
        P'(v) = \frac{P(v \mid x_{1:t-1})}{\sum_{v' \in \mathcal{V}_{\text{min}}} P(v' \mid x_
            {1:t-1})} \quad \text{for } v \in \mathcal{V}_{\text{min}}
    \end{equation}
\end{enumerate}
```

Instruction

```
### **Instruction 1: Implement Weighted Pooling Utility**
* **Functionality Description:** Create a utility function that performs weighted pooling of
    hidden states within fixed-size windows using PyTorch tensor operations. The function
    should aggregate token representations by using a `scatter_add` operation to efficiently
    calculate a weighted sum for each window. It must also generate a new attention mask for
    the resulting pooled sequence, where a pooled position is considered valid if at least
    one token in its original window was valid. This function directly implements the
    summation part of the weighted pooling operation described in Equation 6 of the paper.
* **Expected Function Name:** `fix_window_size_pooling`
* **Input:**
    * `hidden_states` (`torch.Tensor`): The input token representations, with a shape of `(
        batch_size, num_windows, window_size, hidden_size)`.
    * `attention_mask` (`torch.Tensor`): A mask indicating valid tokens in the original, un-
        windowed sequence.
    * `weights` (`torch.Tensor`): The weights to apply to each token representation before
        summing. Must have the same shape as `hidden_states`.
* **Output:**
    * `pooled_hidden_states` (`torch.Tensor`): The resulting sequence of pooled
        representations, with a shape of `(batch_size, num_windows, hidden_size)`.
    * `pooled_attention_mask` (`torch.Tensor`): The new attention mask corresponding to the
        pooled sequence.
```

Figure 6: A example of paper description and code generation instruction

CANONICAL CODE

```python
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> torch.
     FloatTensor:
    probs = torch.softmax(scores, dim=-1)
    top_probs, _ = probs.max(dim=-1, keepdim=True)
    scaled_min_p = self.min_p * top_probs
    tokens_to_remove = probs < scaled_min_p
    sorted_indices = torch.argsort(scores, descending=True, dim=-1)
    # Reorder the `tokens_to_remove` mask to align with the sorted tokens.
    sorted_indices_to_remove = torch.gather(tokens_to_remove, dim=-1, index=sorted_indices)
    sorted_indices_to_remove[..., : self.min_tokens_to_keep] = False
    indices_to_remove = sorted_indices_to_remove.scatter(1, sorted_indices,
         sorted_indices_to_remove)
    scores_processed = scores.masked_fill(indices_to_remove, self.filter_value)
    return scores_processed
```

Figure 7: Example of a generated code

Generated Code

```python
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> torch.
     FloatTensor:
    if not torch.is_tensor(scores):
        raise TypeError("scores must be a torch.Tensor")
    if scores.dim() != 2:
        raise ValueError(f"scores must be a 2D tensor of shape (batch_size, vocab_size), got
             shape {scores.shape}")

    device = scores.device
    dtype = scores.dtype
    batch_size, vocab_size = scores.shape
    probs = torch.softmax(scores, dim=-1)  # shape (batch_size, vocab_size)
    p_max, _ = probs.max(dim=-1, keepdim=True)
    p_scaled = p_max * float(self.min_p)  # shape (batch, 1)
    tokens_to_remove = probs < p_scaled  # (batch, vocab), boolean
    k = min(self.min_tokens_to_keep, vocab_size)
    if k > 0:
        kept_counts = (~tokens_to_remove).sum(dim=-1)  # (batch,)
        need_fix = kept_counts < k  # (batch,)

        if need_fix.any():
            topk_vals, topk_indices = torch.topk(probs, k=k, dim=-1)  # (batch, k)
            keep_topk_mask = torch.zeros_like(tokens_to_remove, dtype=torch.bool, device=
                 device)
            keep_topk_mask.scatter_(1, topk_indices, True)  # (batch, vocab)

            need_fix_idx = need_fix.nonzero(as_tuple=False).squeeze(-1)
            if need_fix_idx.numel() == 1:
                idx = need_fix_idx.item()
                # Un-remove top-k positions for this sequence
                tokens_to_remove[idx] &= ~keep_topk_mask[idx]
            else:
                # If multiple sequences need fixing, use advanced indexing
                tokens_to_remove[need_fix_idx] &= ~keep_topk_mask[need_fix_idx]

    processed_scores = scores.clone()
    if self.filter_value == float("-inf"):
        fill_val = float("-inf")
    else:
        fill_val = float(self.filter_value)

    processed_scores = processed_scores.masked_fill(tokens_to_remove, fill_val)

    return processed_scores
```

Figure 8: Example of a generated code

Feedback

~~~
# Feedback Report
## Feedback 1
... ...

## Feedback 2
**Interface:** __call__
### Description
`min_tokens_to_keep` logic deviates from the canonical sort/gather/scatter approach and uses a
      conditional top-k mask, which can lead to discrepancies in which tokens are preserved

### Analysis
The canonical code enforces the `min_tokens_to_keep` safeguard by sorting the logits,
     gathering the removal mask, unmarking the top-k positions, and scattering back to the
     original order. This approach is guaranteed to mirror the paper's requirement in all tie
     or batch cases. The generated code's manual `topk` + boolean indexing branch may select
     different tokens when probabilities tie or in edge cases, causing mismatches (as seen in
     `test_min_tokens_to_keep`).

### Actionable Feedback
Replace the existing `min_tokens_to_keep` block with the canonical pattern: sort logits,
     gather the removal mask, reset the first `min_tokens_to_keep` entries, and scatter back
     to the original vocabulary order.

### Direct Code Feedback
```text
In `__call__`, locate the section starting with the comment `# Enforce min_tokens_to_keep:`
     and replace it entirely with:
```python
        # ### Implementation Details: ensure at least min_tokens_to_keep tokens are kept
        sorted_indices = torch.argsort(scores, descending=True, dim=-1)
        sorted_mask = torch.gather(tokens_to_remove, dim=-1, index=sorted_indices)
        sorted_mask[..., : self.min_tokens_to_keep] = False
        tokens_to_remove = sorted_mask.scatter(1, sorted_indices, sorted_mask)
~~~
This aligns the implementation with the canonical scatter-based method.
```
```

Figure 9: Feedback to the generated code.

Result Code

```python
def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor) -> torch.
     FloatTensor:

    if not torch.is_tensor(scores):
        raise TypeError("scores must be a torch.Tensor")
    if scores.dim() != 2:
        raise ValueError(f"scores must be a 2D tensor of shape (batch_size, vocab_size), got
             shape {scores.shape}")

    device = scores.device
    batch_size, vocab_size = scores.shape
    probs = torch.softmax(scores, dim=-1)  # shape (batch_size, vocab_size)
    p_max, _ = probs.max(dim=-1, keepdim=True)  # (batch, 1)
    p_scaled = p_max * float(self.min_p)  # shape (batch, 1)
    tokens_to_remove = probs < p_scaled  # (batch, vocab), boolean
    min_keep = min(self.min_tokens_to_keep, vocab_size)
    if min_keep > 0:
        sorted_indices = torch.argsort(scores, descending=True, dim=-1)  # (batch, vocab)
        sorted_mask = torch.gather(tokens_to_remove, dim=-1, index=sorted_indices)  # (batch,
             vocab)
        sorted_mask[:, :min_keep] = False
        new_tokens_to_remove = torch.ones_like(tokens_to_remove, dtype=torch.bool, device=
             device)
        new_tokens_to_remove.scatter_(1, sorted_indices, sorted_mask)
        tokens_to_remove = new_tokens_to_remove
    processed_scores = scores.clone()
    processed_scores = processed_scores.masked_fill(tokens_to_remove, float(self.filter_value)
         )

    return processed_scores
```

Figure 10: Example of revised code.

M PROMPTS

M.1 CODE GENERATION PROMPT

System Prompt

~~~
You are an expert machine learning researcher.
Task description: Your task is to generate high-quality, well-documented, and contextually
     relevant code implementations based on the provided LaTeX version of a research paper and
      a specific user instruction. You will also have access to the existing code repository
     to ensure your generated code integrates seamlessly.
Commands actions:
============= CODE READ TOOL =============
You also have access to a code reading tool.
This tool allows you to read the content of any file in the current repository. It helps you
     understand the codebase context before making changes or generating new code.
You can read a file using the following command:
READ <file path to be read>
READ is the word READ, and <file path to be read> is the relative path of the file you want to
      inspect. This command will return the full content of the specified file.
Use this command when you need to examine the contents of any file in the repository,
     including the current target file.

============= CODE RETRIEVE TOOL =============
You also have access to a code retrieval tool.
This tool allows you to retrieve the most relevant code functions for a given natural language
      query. It helps you understand where and how specific functionality is implemented in
     the codebase.
You can retrieve code using the following command:
RETRIEVE
QUERY
RETRIEVE is the word RETRIEVE, and QUERY is your request in natural language. The response
     will return the top relevant functions, including their code and location in the
     repository. This will be the primary way to locate and explore code related to specific
     functionality or concepts.
Use this command when you want to investigate or modify code related to a particular feature,
     action, or behavior.

============= REWRITE CODE EDITING TOOL =============
You also have access to a code replacing tool.
This tool allows you to entirely re-write/replace all of the current code and erase all
     existing code.
You can use this tool via the following command: ```REPLACE
<code here>
```, where REPLACE is the word REPLACE and <code here> will be the new code that is replacing
     the entire set of old code. This tool is useful if you want to make very significant
     changes, such as entirely changing the code file content. Try limiting the use of
     rewriting and aim for editing the code more.
============= CODE SUBMIT TOOL =============
You also have access to a code submission tool.
This tool executes the current code in the target file and returns the results of its
     execution, including any unit test outcomes and feedback related to the code's behavior.
You can submit code using the following command:
SUBMIT
SUBMIT is the word SUBMIT, and it will run the code currently present in the target file.
     After execution, you will receive the results of any unit tests, along with diagnostic
     messages or errors that occurred during runtime.

============= REPO BROWSE TOOL =============
You also have access to a repository browsing tool.
This tool allows you to browse the entire code repository associated with the current task. It
      helps you understand the overall structure, locate files, and explore code across
     different modules or components.
You can browse the repository using the following command:
BROWSE
BROWSE is the word BROWSE, and it will return a list of files and directories in the
     repository. This will be useful for understanding how different parts of the codebase are
      organized and where specific functionality is implemented.

You should use these actions to access the code file and reterieve the code repository
     information. Before the actions you should relect about the context and make sure the
     command is following the correct syntex.
You should follow this format:
reflect: [Your reflection on the context and the action you are going to take]
action:
[The action you are going to take]
Your reflect should cover these aspects:

1. Execution Results
~~~

```

- Diagnose compilation or runtime errors (e.g., syntax errors, missing dependencies).

- Inspect test case outcomes, focusing on which tests failed and the corresponding error
    messages.

- Consider performance-related signals if available (e.g., timeouts, memory overuse).

2 Code Consistency

- Check alignment between the generated code and the method description in the paper.

- Ensure compatibility with the existing repository (function signatures, class structures,
    module dependencies).

- Maintain coherent structure and style consistent with the project.

3. Feedback Integration

- Extract actionable guidance from human (or simulated) feedback.

- Identify logical flaws, missing components, or suggested improvements.

- Translate natural-language feedback into concrete modification strategies.

4. History Awareness

- Review previous attempts to avoid repeating failed solutions.

- Identify patterns in past mistakes and refine strategy accordingly.

5. Next-Step Planning

- Identify the current code status.

- Decide on the priority of actions (e.g., reading files, searching the repository, or editing
     code).

- Determine the scope of changes (minor patch vs. major refactor).

- Identify additional information needs before generation.
```

User prompt

```
'''## Research Code Generation Request
---
**1. Relevant LaTeX Content:**
---
Below you will find the necessary information to generate the requested code. Please process
     all sections carefully.
{latex code}

---
2. Code Generation Instruction:
---
{code generation instruction}

---
3. Conversation History
---
{conversation history}

---
4. Current Code implementation
---
{current code content}

---
5. Feedback on Previous Submition
---
{feedback content}

---
6. Action Execution Result
---
{action execution result}

'''
```

M.2 Feedback Generation Prompt

System Prompt

```
You are an expert Code Analysis Agent. Your task is to generate detailed and actionable
     feedback on a piece of generated code. This feedback will be used to improve future code
     generation attempts.

You will be provided with the following information to perform your analysis:

1.  **LaTeX Code from Research Paper:** This document (or snippet) describes the intended
     mathematical or algorithmic functionality that the target code should implement. Use this
      to understand the core logic, equations, and theoretical underpinnings.
2.  **User Instruction/Prompt:** This is the original instruction given to the code generation
      model that produced the 'Generated Code'. Evaluate if the generated code aligned with
     this instruction.
3.  **Canonical Code with Comments:** This is a reference or ideal implementation of the
     desired functionality. It contains specific comments highlighting key aspects, logic
     flows, or potential pitfalls. Use this as a reference for feedback.
4.  **Generated Code:** This is the code produced by another LLM that you need to analyze.
5.  **Error Information:** This is the execution error information of the generated
     information.
6.  **Generation Guidance & Feedback Specification:** This document outlines:
    * **Generation Guidance:** The generation format for this task.
    * **Feedback Categories:** The predefined categories your feedback should address.
    * **Feedback Granularity:** The level of detail required for your feedback.

Based on the provided information, you must generate your feedback based on the user
     specification.

# **Your Task:**

Carefully compare the **LLM-Generated Code** and the **Canonical Code** for the specified **
     interface, function, or method** as instructed in the **Code Generation Instruction**.

Your tasks are as follows:

1. **Identify all differences:**
   Explicitly list every relevant difference between the **LLM-Generated Code** and the **
        Canonical Code** within the instructed interface, function, or method. For each
        difference, clearly specify what is present in one version and absent or different in
        the other.

2. **Analyze each difference:**
  Clearly explain the impact of the difference and why it lead to an error in the context of
       the instruction and the canonical solution (expected code).

2. **Analyze each difference:**
    For each difference, analyze its significance and determine whether it affects correctness
         , completeness, style, or performance. Select the most appropriate feedback category
          (T0--T4) for each difference. Clearly explain the impact of the difference and why it
           lead to an error in the context of the instruction and the canonical solution (
          expected code).


4. **Direct code-level feedback:**
   For each actionable item, provide a detailed description of exactly how to modify the LLM-
        Generated Code to resolve the difference. The code should be identical with the code
        snippet .

**Output Format:**
Produce a JSON object with a `"differences"` array. For each difference, include the following
      fields:
- `"interface"`: The name of the interface, function, or method where the difference occurs
- `"category"`: The feedback category (T0-T4)
- `"description"`: A brief description of the difference
- `"analysis"`: How current implementation lead to an error, why it is not correct
- `"actionable_feedback"`: Clear, concrete, and actionable guidance for correction
- `"direct_code_feedback"`: Consistent with the actionable feedback, a detailed description of
      how to modify the code to resolve the difference. Use the canonical code snippet as the
     guidance feedback.

**Example Output:**
{{
  "differences": [
    {{
      "interface": "calculate_total",
      "category": "T1",
      "description": "The LLM-Generated Code does not initialize the variable 'var' before use
           .",
      "analysis": "The generated code omits the initialization of the variable 'var', which
           can cause a runtime error or incorrect results.",
      "actionable_feedback": "Ensure that all variables are properly initialized before they
           are used.",
      "direct_code_feedback": "Add the line `var = []` before the iteration to initialize the
           variable as shown in the Canonical Code."
    }}
    // Add more differences as needed
  ]
}}
Instructions:

* Focus on all differences in the instructed code region, not just those leading to major
     errors.

* Differences can include changes in logic, missing or additional statements, variable
     initialization, return values, control flow, structure, function signatures, etc.

* Be explicit and systematic for each observed difference.
```

User Prompt

~~~
**Task Information:**

Please generate code analysis feedback based on the following information:

1.  **Paper Description (LaTeX):**
    ```latex
    {latex code}
    ~~~

2. **Code Generation Instruction:**
    ```
    {code generation instruction}
    ```

3. **Canonical Code (Ground Truth/Example):**
    ```{}
    {canoncial code}
    ```

4. **LLM-Generated Code:**
    ```{}
    {generated code content}
    ```
5. **Error Information:**
    ```{execution error log}
    ```

6. **Generation Guidance & Feedback Specification:**
    Subject: Understanding the 5 Feedback Categories for Code Generation
To ensure we evaluate LLM-generated code consistently, we use a structured feedback system.
     This system helps us pinpoint the exact nature of an error.

Our system is based on two simple questions:
1. **What kind of error is it?** (This is the **Category**, T_0-T_4)
2. **How much help do we provide?** (This is the **Level**, L_0-L_4)

This document introduces the five **Categories** of feedback. When analyzing a piece of code,
     your first step is to identify which of these five categories the *most significant*
     error falls into.

The 5 Feedback Categories ($T_0 - T_4$)

Here is a guide to each category, designed to help you quickly classify any error you
     encounter.

T_0: Code Structure (Planning) Feedback
* **In a Nutshell:** The overall architectural plan is wrong.
* **Core Question:** Is the code's high-level organization or structure fundamentally
     different from the intended design, even if some of the internal logic is correct?
* **Look for:**
    * A single, monolithic function when it should have been broken down into smaller helper
         methods.
    * A class that is missing essential methods required for its core purpose.
    * Incorrect data flow or a flawed high-level implementation strategy.

T_1: Code Correctness Feedback
```
```

```
* **In a Nutshell:** The code is fundamentally broken and won't run.
* **Core Question:** Does the code fail due to a basic syntax mistake, a typo, or a
     fundamental Python error?
* **Look for:**
    * `SyntaxError`: e.g., an unclosed parenthesis or unterminated string.
    * `NameError`: e.g., using a variable before it has been assigned.
    * `TypeError`: e.g., trying to add a string to an integer (when not part of the core
         algorithm's logic).

#### **$T_2$: Implementation Alignment Feedback**
* **In a Nutshell:** The code ignores or misinterprets the provided paper or instructions.
* **Core Question:** Can I point to a specific sentence, formula, or requirement in the
     provided text that this code directly contradicts?
* **Look for:**
    * Implementing Equation (4) from the paper when Equation (3) was specified.
    * Using the wrong parameter names in a function call compared to the instructions.
    * Producing an output with the wrong shape or data type described in the paper.

#### **$T_3$: Knowledge & Context Feedback**
* **In a Nutshell:** The code fails because it's missing crucial knowledge that was **not**
     provided.
* **Core Question:** Is the fix something the LLM would need to know from general domain
     expertise, or is it an implicit "secret" of the original codebase that wasn't written
     down?
* **Look for:**
    * Using placeholder logic for a complex but standard operation (e.g., "TODO: implement
         topology calculation here").
    * Needing a specific, unstated hyperparameter (e.g., a learning rate of `0.001`).
    * Using a generic library when a specific, domain-standard library (e.g., `gudhi`, `
         huggingface`) is the obvious choice.

#### **$T_4$: Repository Integration Feedback**
* **In a Nutshell:** The code reinvents the wheel or fails to use existing tools from the
     codebase.
* **Core Question:** Did the code rewrite a helper function or class that was already
     available in the project's existing files?
* **Look for:**
    * Writing a new `normalize_text` function when `utils.text.normalize()` already exists.
    * Incorrectly using a provided class from another module in the repository.
    * Ignoring the established coding style or conventions of the repository.

---

### How to Choose the Right Category: A Decision Guide

Always start at the top of this list. The first "Yes" determines the error's category. This
     helps distinguish between similar issues (especially $T_2$ and $T_3$).

1.  **Is it a basic syntax/runtime error?**
    * **Yes?** -> It's **$T_1$**.

2.  **No? Okay, does it ignore or re-implement existing code from the repo?**
    * **Yes?** -> It's **$T_4$**.

3.  **No? Is the overall code architecture or structure the main problem?**
    * **Yes?** -> It's **$T_0$**.

4.  **No? Can I find the fix *explicitly written* in the paper/instructions?**
    * **Yes?** -> It's **$T_2$**. (The LLM failed to read carefully).

5.  **No? Is the fix based on knowledge *outside* the paper/instructions?**
    * **Yes?** -> It's **$T_3$**. (The LLM lacked necessary context).

Please generate the feedback with the specified format.
```

M.3 Category Classification Prompt

System Prompt

```
You are an expert code reviewer and research engineer.
Your role:
- Classify feedback about code into error categories(T0-T4).
- Judge whether the next version of the code (v2) adopted the feedback and whether the issue
     was resolved.
- Follow definitions and disambiguation strictly.
- Provide concise evidence (<=3 items) with location references.
- If unsure, use `"uncertain"`.

```

```
## Input Specification
The user prompt will provide the following blocks of content (each delimited clearly):

1. **[PAPER]** - Excerpts from the research paper (may include formulas, equations, section
     refs).
2. **[INSTRUCTION]** - The instruction given to generate code based on the paper.
3. **[GROUND_TRUTH_CODE]**(optional) - A reference or canonical code implementation.
4. **[V1_CODE]** - The first generated code version.
5. **[V1_RUN_LOG]** - Execution results and error logs for v1.
6. **[FEEDBACK]** - Feedback on v1 (may contain multiple feedback entries).
7. **[V2_CODE]** - The new generated code (after applying feedback).
8. **[V2_RUN_LOG]** - Execution results and error logs for v2.

## Error Types
- **T0: Code Structure (Planning)** - The error is because the high-level design, architecture
     , data flow, modularization, or strategy is not aligned with the canonical or intended
     design.
- **T1: Code Correctness** - The error is because of general syntax, runtime, or basic
     programming logic errors.
- **T2: Implementation Alignment** - The error is because the code is implemented with
     misalignment to the paper algorithm, formulas, or instruction I/O requirements.
- **T3: Knowledge & Context** - The error is because of missing domain knowledge, conventions,
      implicit assumptions, or author preference.
- **T4: Repository Integration** - The error is because of not using the same function defined
      within this repo (other file), misuse of existing codebase (from other file), or
     reimplementing helpers.

## Disambiguation Guidelines
- **T1 vs T2**:
  If the issue would arise in any code regardless of the paper -> T1.
  If it stems from not following the paper's algorithm or I/O specs -> T2.

- **T0 vs T2**:
  T0 = structural/architectural guidance, modularization.
  T2 = algorithmic/methodological alignment with the paper.

- **T4 vs T0**:
  T4 = repository-specific (reuse of helpers, placement, conventions from other file).
  T0 = general design/architecture not tied to repo assets.

- **T3 vs Others (Key Rule)**:
  - Assign **T3** when the feedback relies on *external or implicit knowledge, or author
       preference/insights in code* not fully contained in the instruction, repo, or paper.
  - Examples: domain-standard library usage, interpreting ambiguous terms, applying author
       preference/insights or common domain practices, filling in paper omissions.
  - If the feedback can be resolved solely by:
    - fixing syntax/runtime -> T1,
    - aligning with explicit paper details or instruction specification -> T2,
    - restructuring modules/classes -> T0,
    - reusing repo assets -> T4.
  - Otherwise, if it depends on **domain expertise or implicit assumptions**, classify as T3.

## Evaluation Pipeline Specification
When analyzing inputs, follow these ordered steps:

1. **Parse Input**
   - Collect paper excerpts, code generation instruction, ground truth (if any), v1 code &
        logs, feedback text, and v2 code & logs.
   - Split multi-point feedback into atomic items.

2. **Classify Error Type (T0-T4)**
   Use the following decision rules:
   - **T0 (Structure/Planning)**: Feedback is about *how code is organized or architected*, e.
        g., "this logic should be a class method," "split into functions," "refactor data flow
        ."
   - **T1 (Correctness)**: Feedback is about *generic programming bugs* like syntax errors,
        type mismatches, variable misuse, bad loop conditions, out-of-bounds. No relation to
        paper content.
   - **T2 (Implementation Alignment)**: Feedback is about *not matching the paper or task spec
        *, e.g., wrong formula, wrong output shape vs. paper, incorrect algorithm step.
   - **T3 (Knowledge & Context)**: Feedback requires *domain knowledge, implicit assumptions,
        or conventions* to understand, e.g., "use library.function() for NLP task," "by '
        attention' they mean Scaled Dot-Product," "paper omits activation X."
   - **T4 (Repository Integration)**: Feedback is about *using or misusing existing repo code
        *, e.g., "use utils.DataLoader instead of reimplementing," "place logic in Model.
        forward," "use config object."

3. **Judge Adoption (adopted)**
   - Compare v1 and v2.
   - If v2 reflects meaningful changes related to the feedback -> `YES`.
   - If no change -> `NO`.
   - If partially implemented -> `PARTIAL`.

4. **Judge Resolution (resolved)**
   - Check v2 logs, outputs, and alignment with paper/instructions.
   - If the problem is fully fixed -> `YES`.
   - If still present -> `NO`.
   - If partially improved -> `PARTIAL`.

5. **Explain Decision**
   - Write a concise explanation (1-3 sentences) citing evidence (e.g., code diff, log line,
        paper reference).

6. **Confidence Score**
   - Assign a float 0-1 reflecting certainty in classification and judgments.
   - Higher = more certain, lower = less certain.

7. **Output**
   - Return results as the specified output format
    error_type: ErrorType = Field(..., description="T0-T4")
    adopted: AdoptStatus = Field(..., description="YES/NO/PARTIAL")
    resolved: AdoptStatus = Field(..., description="YES/NO/PARTIAL")
    explain_error_type: str = Field(..., min_length=1, max_length=500, description="Describe
         the reason the error is categorized to this error type.")
    explain_adopted_solved: str = Field(..., min_length=1, max_length=500, description="
         Describe the reason the feedback is adopted or not and the error is resolved")
    confidence_score: confloat(ge=0.0, le=1.0) = Field(..., description="0-1")



## Important Notes:
- **The canonical code meaning in feedback** mentioned in the feedback is the expcted
     generated code(is the GROUND_TRUTH_CODE in the user prompt) for the task, but **is not
     the repo code**. You need to analysis the root cause of the error and catecory to the
     correct category.
- **Output content**:
    If the error is in category 1, the explain_error_type should explain what syntex error or
         runtime error os.
    If the error is in category 2, the explain_error_type should explain how the **instruction
          or paper** is specified and how the code is misaligned with the paper description or
          specification.
    If the error is in category 3, the explain_error_type should explain how the code is
         implemented and what information is not specified in the **paper or instruction**.
    If the error is in category 4, the explain_error_type should explain what and how the
         function that defined in the repository is misused or not used as it expected.
'''
```

User Prompt

```
You are given the following data.
Please analyze and output structured JSON according to the schema above.


[PAPER]
{paper content}

[INSTRUCTION]
{code generation instruction}

[GROUND_TRUTH_CODE]
{ground truth(optional)}

[V1_CODE]
{generated code v1}

[V1_RUN_LOG]
{v1 code execution log}

[FEEDBACK]
{feedback contents}

[V2_CODE]
{revised code v2}

[V2_RUN_LOG]
{v2 code execution log}

## Important Notes:
- **The canonical code meaning in feedback** mentioned in the feedback is the expcted
     generated code(is the GROUND_TRUTH_CODE in the user prompt) for the task, but not some
```

```
    code exisits in the repo. You need to analysis the root cause of the error and catecory
    to the correct category.
```

N Annotation Example

N.1 Paper description

```
\section{Problem formulations}
\label{sec:background}
\subsection{Setting}
We focus on statistical inference problems for the parameter \(\theta^{*} \equiv \theta^{*}(\
    mathbb{P}) \in \RR^K\) defined on the joint distribution of \((\X, Y) \sim \mathbb{P}\),
    where \(Y \in \mathcal{Y}\) is a scalar outcome and \(\X \in \mathcal{X}\) be a \(K\)-
    dimensional vector representing features. We are interested in estimating \(\theta^{*}\)
    using labeled data \(\calL = \{(\X_i, Y_i), i = 1,\cdots,n\} \equiv (\XL, \YL)\),
    unlabeled data \ \(\calU = \{\X_i, i = n+1,\cdots,n+N\} \equiv \XU\), and a pre-trained
    ML model \(\wh f(\cdot): \mathcal{X} \rightarrow \mathcal{Y}\). Here, \(f(\cdot)\) is a
    black-box function with unknown operating characteristics and can be mis-specified. We
    also require an algorithm \(\cA\) that inputs the labeled data \(\calL\) and returns a
    consistent and asymptotically normally distributed estimator \(\widehat{\theta}\) for \(\
    theta^*\). There are three common ways in the literature to estimate \(\theta^*\):
\begin{itemize}
    \item \textbf{Classical statistical methods} apply algorithm \(\cA\) to only labeled data
        \(\calL = (\XL, \YL)\), and returns the estimator and its estimated variance \([\wh\
        bt_\calL, \wh\myVar(\wh\bt_\calL)]\). Valid confidence intervals and hypothesis tests
         can then be constructed using the asymptotic distribution of the estimator. However,
         it ignores the unlabeled data and ML prediction.

    \item \textbf{Imputation-based methods} treat ML prediction \(\wh f\) in the unlabeled
        data as the observed outcome, and apply algorithm \(\cA\) to \(\calU = (\XU, \fU)\).
        We denote the estimator and estimated variance as \([\wh\bmeta_\calU, \wh\myVar(\wh\
        bmeta_\calU)]\). This has been shown to give invalid inference results and false
        scientific findings \citep{wang2020methods,angelopoulos2023prediction,
        miao2023assumption,miao2024valid}.
    \item  \textbf{ML-assisted inference methods} use both \(\calL\) and \(\calU\) as input.
        These approaches add a debiasing term in the loss function (or estimating equation)
        for M-estimation problems, thus removing the bias from the imputation-based
        estimators and producing results that are statistically valid and universally more
        powerful compared to classical methods \citep{angelopoulos2023ppi++,
        miao2023assumption,miao2024valid}.
\end{itemize}
Next, we use an example to provide intuition on ML-assisted inference and our protocol.

\subsection{Building the intuition with mean estimation}
We consider the mean estimation problem, where \(\theta^* = \E[Y_i] \equiv \argmin_\theta\E[\
    frac{1}{2}\left(Y_i-\theta\right)^2]\). The classical method only takes the labeled data
    \(\YL\) as input and yields an unbiased and consistent estimator for \(\theta^*\):
\(
    \wh\theta_\calL = \argmin_\theta  \meann \frac{1}{2}\left(Y_i-\theta\right)^2 = \meann Y_i
\). The imputation-based method only takes the unlabeled data \(\fU\) as input and returns
\(
    \wh\eta_\calU = \argmin_\theta  \frac{1}{N} \sum_{i=n+1}^{n+N} \frac{1}{2}(\wh f_i-\theta)
        ^2 = \frac{1}{N} \sum_{i=n+1}^{n+N} \wh f_i
\). It is a biased and inconsistent estimator for \(\E[Y_i]\) if the ML model \(\wh f\) is mis
    -specified. To address this, ML-assisted estimator takes both labeled data \((\YL, \fL)\)
     and unlabeled data \(\fU\) as input and adds a debiasing term to the loss function to
    rectify the bias caused by ML imputation \citep{angelopoulos2023prediction,
    miao2023assumption,miao2024valid,gan2023prediction}:
\begin{small}
\begin{align*}
    \wh\theta_{\texttt{MLA}}
    &=
    \argmin_\theta \frac{1}{2}
    \{
    \wh\omega_0\meanN (\wh f_i-\theta)^2
    -
    \underbrace{[
    \wh\omega_0\meann (\wh f_i-\theta)^2
    -
    \meann \left(Y_i-\theta\right)^2
    ]}_{\text{Debiasing term}}
    \}\\
    &=
    \wh\omega_0\meanN\wh f_i -
     \underbrace{
     [\wh\omega_0\meann \wh f_i -\meann y_i]
     }_{\text{Debiasing term}},
\end{align*}
\end{small}
where the modified loss ensures the consistency of the ML-assisted estimator and  the weight \(\wh\omega_0 = \frac{\wh\covl[Y, \wh{f}]/n}{\wh\varl[\wh{f}]/n + \wh\varu[\wh{f}]/N}\) ensures that ML-assisted estimator is no less efficient than the classical estimator with arbitrary ML predictions:
\(
\myVar(\wh\theta_{\texttt{MLA}})
=
\myVar(\wh\theta_\calL)
-
\frac{\operatorname{Cov}[Y, \widehat{f}]}{n \operatorname{Var}[\widehat{f}]+n^2 \operatorname{Var}[\widehat{f}] / N}
\leq
\myVar(\wh\theta_\calL) \).

Our proposed method is motivated by the observation that the \textbf{sufficient statistics} of the ML-assisted estimator \(\wh\theta_{\texttt{MLA}}\) and its estimated variance \(\wh\myVar(\wh\theta_{\texttt{MLA}})\) are the following summary statistics:
\[
    \wh \bt_{\texttt{ss}}
    =
    (\meann y_i, \meann \wh f_i, \meanN \wh f_i)
    \text{ and }
    \wh\myVar(\wh \bt_{\texttt{ss}})
    =
    \begin{bmatrix}
    \wh\varl[Y]/n & \wh\covl[Y, \wh{f}]/n & 0\\
    \wh\covl[Y, \wh{f}]/n & \wh\varl[\wh{f}]/n & 0 \\
    0 & 0 & \wh\varu[\wh{f}]/N \\
    \end{bmatrix}
\]

Moreover, they can be easily obtained by applying the \textbf{same algorithm} \(\cA\) (mean estimation) to
\begin{itemize}[itemsep=1pt, parsep=0pt]
    \item labeled data with observed outcome \(
    \cA(\YL)
    \rightarrow [\wh\theta_{\calL}, \wh\myVar(\wh\theta_{\calL})]
    =
    [\meann y_i, \wh\varl[Y]/n]
    \)
    \item labeled data with predicted outcome \(
    \cA(\fL)
    \rightarrow
    [\wh\eta_{\calL}, \wh\myVar(\wh\eta_{\calL})] =
    [\meann \wh f_i, \wh\varl[\wh{f}]/n]
    \)
    \item unlabeled data with predicted outcome\(\cA(\fU)  \rightarrow  [\wh\eta_{\calU},\wh\myVar(\wh\eta_{\calU})]
    =
    [\meanN \wh f_i, \wh\varu[\wh{f}]/N]
    \)
    \item bootstrap of labeled data \(\cA[(\YL, \fL)_q, q = 1, \dots, Q]\) for estimation of \(\wh\myCov(\wh\theta_{\calL}, \wh\eta_{\calL}) = \wh\covl[Y, \wh{f}]/n\). Here, \((\YL, \fL)_q\) represents the \(q\)-th bootstrap of labeled data.
\end{itemize}

Combining these summary statistics for one-step debiasing \(\wh\omega_0\wh\eta_{\calU} - (\wh\omega_0\wh\eta_{\calL} - \wh\theta_{\calL})\) recovers \(\wh\theta_{\texttt{MLA}}\).

To summarize, an algorithm for mean estimation, coupled with resampling, is sufficient for ML-assisted mean estimation. This observation inspired us to generalize this protocol for a broad range of tasks. Our protocol illustrated in \cref{fig:workflow} only requires three steps: 1) using a pre-trained ML model to predict outcomes for labeled and unlabeled data, 2) applying existing analysis routines to generate summary statistics, and 3) using these statistics in a debiasing procedure to produce statistically valid results in ML-assisted inference.

\begin{figure}[H]
    \centering
    \includegraphics[width = 0.85\linewidth]{Figures/PSPS_workflow.pdf}
    \caption{Workflow of \PSPS~for Task-Agnostic ML-Assisted Inference.}
    \label{fig:workflow}
\end{figure}


\section{Methods}
\label{sec:method}
\subsection{General protocol for \PSPS}
```

```
Building on \cref{sec:background}, we formalized our protocol in \cref{fig:workflow} for ML-
     assisted inference:
\begin{algorithm}[H]
  \caption{\PSPS~for ML-assisted inference}
  \label{alg:psps}
  \begin{algorithmic}[1]
  \Require A pre-trained ML model \(\wh f\), labeled data \(\calL = (\XL, \YL)\), unlabeled
       data \(\calU = \XU\)
  \State Use the ML model \(\wh f\) to predict the outcome in both labeled and unlabeled data.
  \State Apply the algorithm \(\cA\) in the analysis routine to
  \begin{itemize}
      \item labeled data \((\XL, \YL)\) and obtain \([\wh \bt_\calL, \wh\myVar(\wh \bt_\calL)
           ]\)
      \item labeled data \((\XL, \fL)\) and obtain \([\wh \bmeta_\calL, \wh\myVar(\wh \bmeta_\
           calL)]\)
      \item unlabeled data with \((\XU, \fU)\) and obtain \([\wh \bmeta_\calU, \wh\myVar(\wh \
           bmeta_\calU)]\)
      \item \(Q\) bootstrap of labeled data \((\XL, \YL, \fL)_q, q = 1,\dots,Q\) and obtain
           \(\wh\myCov(\wh \bt_\calL, \wh \bmeta_\calL)\).
  \end{itemize}
  \State Employ one-step debiasing to the summary statistics in step2:
  \begin{center}
  \(
      \btpsps = \wh\bomega_0\trans\wh{\bmeta}_{\calU} -  (\wh\bomega_0\trans\wh{\bmeta}_{\calL
           } - \wh{\bt}_{\calL}),
  \)
  \end{center}
  where \(\wh\bomega_0 = [\wh\myVar(\wh \bmeta_\calL) + \wh\myVar(\wh \bmeta_\calU)]^{-1}\wh\
       myCov(\wh \bt_\calL, \wh \bmeta_\calL)\) and \(\wh\myVar(\btpsps) = \wh\myVar(\wh \bt_\
       calL) - \wh\myCov(\wh \bt_\calL, \wh \bmeta_\calL)\trans[ \wh\myVar(\wh \bmeta_\calL) +
        \wh\myVar(\wh \bmeta_\calU)]^{-1}\wh\myCov(\wh \bt_\calL, \wh \bmeta_\calL)\)
  \Ensure ML-assisted point estimator \(\btpsps\), standard error \(\sqrt{\wh\myVar(\btpsps)
       }\), \(\alpha\)-level confidence interval for the \(k\)-th coordinate \( \mathcal{C}_{\
       alpha,k}^{\PSPS}=(\wh{\bt}_{\PSPS_k} \pm z_{1-\alpha / 2} \sqrt{\wh\myVar(\btpsps)_{kk
       }})\), and (two-sided) p-value \( 2 ( 1 - \Phi(| \frac{\wh{\bt}_{\PSPS_k}}{\sqrt{\wh\
       myVar(\btpsps)_{kk}}}} | ) )\), where \(\Phi\) is the CDF of the standard normal
       distribution.
   \end{algorithmic}
\end{algorithm}

The only requirements for our protocol are: i) algorithm \(\cA\), when applied to labeled data
      \((\bX_\calL, Y_\calL)\), returns a consistent and asymptotically normally distributed
     estimator of \(\bt^*\); ii) labeled and unlabeled data are independent and identically
     distributed. Under these assumptions, the summary statistics have the following
     asymptotic properties:
\begin{align}
\label{eq:esteq}
n^{1 / 2}\left(\begin{array}{c}
\wh{\bt}_{\calL} - \bt^* \\
\wh{\bmeta}_{\calL} - \bmeta \\
\wh{\bmeta}_{\calU} - \bmeta \\
\end{array}\right) \coD \mathcal{N}\left\{
\left(\begin{array}{c}
\0_K \\
\0_K \\
\0_K
\end{array}\right)
,
\left(\begin{array}{ccc}
\V(\wh \bt_\calL)& \V(\wh \bt_\calL, \wh \bmeta_\calL) & \0\\
\V(\wh \bt_\calL, \wh \bmeta_\calL) &  \V(\wh \bmeta_\calL) & \0 \\
\0 & \0 &  \rho\V(\wh \bmeta_\calU)
\end{array}\right)
\right\},
\end{align}

where \(\bmeta \equiv \bmeta(\mathbb{P}_{\wh f}) \in \RR^K\) is defined on \((\X, \wh f) \sim
     \mathbb{P}_{\wh f}\), \(\V(\cdot)\) denotes the asymptotic variance and covariance of a
     estimator, and \(\rho = \frac{n}{N}\). The asymptotic approximation gives
\(\V(\wh \bt_\calL) \approx n\myVar(\wh \bt_\calL), \V(\wh \bt_\calL, \wh \bmeta_\calL) \
     approx n\myCov(\wh \bt_\calL, \wh \bmeta_\calL), \V(\wh \bmeta_\calL) \approx n\myVar(\wh
      \bmeta_\calL)\) and \(\V(\wh \bmeta_\calU) \approx N\myVar(\wh \bmeta_\calU)\). Here, we
      do not require \(\wh \bmeta_\calL\) and \(\wh \bmeta_\calU\) to be consistent for \(\bt
     ^*\), thus allows arbitrary ML model.

With the summary statistics following a multivariate normal distribution asymptotically, the
     debiased estimator \(\btpsps = \wh\bomega_0\trans\wh{\bmeta}_{\calU} -  (\wh\bomega_0\
     trans\wh{\bmeta}_{\calL} - \wh{\bt}_{\calL})\) is consistent for \(\theta^*\) and
     asymptotically normally distributed (\cref{thm:asynor}). Therefore, by plugging in a
     consistent estimator for its asymptotic variance \(\V(\btpsps) \approx n\myVar(\btpsps)\)
     , valid confidence interval and hypothesis testing can be achieved.
```

```

\begin{remark}
\PSPS~is more "task-agnostic" than existing methods in three aspects:
\begin{enumerate}
    \item For M-estimation tasks, currently, only mean and quantile estimation, as well as
        linear, logistic, and Poisson regression, have been implemented in software tools and
         are ready for immediate application. For other M-estimation tasks, task-specific
        derivation of the ML-assisted loss functions and asymptotic variance via the central
        limit theorem are necessary. After that, researchers still need to develop software
        packages and optimization algorithms to carry out real applications. In contrast, \
        PSPS~only requires already implemented algorithms and software designed for classical
         inference based on labeled data.
    \item For problems that do not fall under M-estimation but have asymptotically normally
        distributed estimators, only \PSPS~can be applied, and all current methods would fail
        . The principles behind ML-assisted M-estimation do not extend to these tasks.
    \item Even for M-estimation tasks that have already been implemented, \PSPS offers the
        additional advantage of relying solely on summary statistics. The "task-specific
        derivations" refer not only to statistical tasks but also to scientific tasks. Real-
        world data analysis in any scientific discipline often involves conventions and
        nuisances that require careful consideration. For example, our work is partly
        motivated by GWAS \citep{uffelmann2021genome}. Statistically, GWAS is a linear
        regression that regresses an outcome on many genetic variants. While the regression-
        based statistical foundation is simple, conducting a valid GWAS requires accounting
        for numerous technical issues, such as sample relatedness (i.e., study participants
        may be genetically related) and population structure (i.e., unrelated individuals of
        the same ancestry are both genetically and phenotypically similar, creating
        confounded associations in GWAS). Sophisticated algorithms and software have been
        developed to address these complex issues \citep{mbatchou2021computationally}. It
        will be very challenging if all these important features need to be reimplemented in
        an ML-assisted GWAS framework. With our \PSPS~protocol, researchers can utilize
        existing tools that are highly optimized for genetic applications to perform ML-
        assisted GWAS. This adaptability is not just limited to GWAS, but is a major feature
        of our approach across scientific domains. \PSPS~enables researchers to conduct ML-
        assisted inference using well-established data analysis routines.
\end{enumerate}
\end{remark}

\begin{remark}
     The "federated data analysis" feature of \PSPS~refers to the fact that we only require
          summary statistics as input for inference, rather than individual-level raw data (
          features $\X$ and label $Y$). For example, consider a scenario where labeled data is
           in one center and unlabeled data is in another, yet researchers cannot access
          individual-level data from both centers simultaneously. Under such conditions,
          current ML-assisted inference, which relies on accessing both labeled and unlabeled
          data to minimize a joint loss function, is not feasible. However, \PSPS~circumvents
          this issue by aggregating summary statistics from multiple centers, thereby
          performing statistical inference while upholding the privacy of individual-level
          data.
\end{remark}

\subsection{Theoretical guarantees}
In this section, we examine the theoretical properties of \(\PSPS\). In what follows, \(\coP\)
      denotes convergence in probability and \(\coD\)  denotes convergence in distribution.
     All proofs are deferred to the \cref{app:proof}.

The first result shows that our proposed estimator is consistent, asymptotically normally
     distributed, and uniformly better in terms of element-wise asymptotic variance compared
     with the classical estimator based on labeled data only.
\begin{theorem}
\label{thm:asynor}
Assuming equation~\eqref{eq:esteq} holds, then \(\btpsps \coP \bt^*\), and
\[
    n^{1 / 2}(\btpsps - \bt^*) \coD
    \mathcal{N}\left(\0, \V(\btpsps)\right),
\]
where \(\V(\btpsps) = \V(\wh \bt_\calL) - \V(\wh \bt_\calL, \wh \bmeta_\calL)\trans( \V(\wh \
     bmeta_\calL) +  \rho\V(\wh \bmeta_\calU))^{-1}\V(\wh \bt_\calL, \wh \bmeta_\calL)\).
     Assume the \(k\)-th column of \(\V(\wh \bt_\calL, \wh \bmeta_\calL)\) is not a zero
     vector and at least one of \(\V(\wh \bmeta_\calL)\) and \(\V(\wh \bmeta_\calU)\) are
     positive definite, then \(\V(\btpsps)_{kk} \leq \V(\wh \bt_\calL)_{kk}\). With \(\wh\V(\
     btpsps) \coP \V(\btpsps), \lim_n \mathbb{P}(\theta_k^{\star} \in \mathcal{C}_{\alpha,k
     }^{\PSPS})=1-\alpha\).
\end{theorem}

\(\wh\V(\btpsps)\) can be obtained by applying the algebraic form of  \(\V(\btpsps)\) using
     the bootstrap estimators for \(\V(\wh \bt_\calL), \V(\wh \bmeta_\calL), \V(\wh \bt_\calL,
      \wh \bmeta_\calL),\) and \(\V(\wh \bmeta_\calU)\). The regularity conditions for
     consistent bootstrap variance estimation are outlined in Theorem 3.10 (i) of \citep{
     shao2012jackknife}. We also refer readers to \citep{hahn2021bootstrap}, which showed that
      bootstrap-based variance provides valid but potentially conservative inference.

```

```
This result indicates that a greater reduction in variance for the ML-assisted estimator is
     associated with larger values of \(\V(\wh \bt_\calL, \wh \bmeta_\calL)\) and smaller
     values of \(\V(\wh \bmeta_\calL)\), \(\V(\wh \bmeta_\calU)\), and \(\rho\). The variance
     reduction term \([\V(\wh \bt_\calL, \wh \bmeta_\calL)\trans( \V(\wh \bmeta_\calL) +  \rho
     \V(\wh \bmeta_\calU))^{-1}\V(\wh \bt_\calL, \wh \bmeta_\calL)]_{kk}\) can also serve as a
      metric for selecting the optimal ML model in ML-assisted inference.

Our next result shows that three existing methods, i.e., \texttt{PPI}, \texttt{PPI++}, and \
     texttt{PSPA}, are asymptotically equivalent to \PSPS~with different weighting matrices. A
      broader class for consistent estimator of \(\bt^*\) is \(\wh{\bt}(\bomega) = \bomega \
     trans\wh{\bmeta}_{\calU} - (\bomega \trans \wh{\bt}_{\calL} - \wh{\bmeta}_{\calL})\),
where \(\bomega\) is a \(K\times K\) matrix. The consistency of \(\wh{\bt}(\bomega)\)  for \(\
     bt^*\) only requires \(\bomega\trans(\wh{\bmeta}_{\calU} - \wh{\bmeta}_{\calL}) \coP \
     mathbf{0}\). Since \((\wh{\bmeta}_{\calU} - \wh{\bmeta}_{\calL}) \coP \mathbf{0}\),
     assigning
arbitrarily fixed weights for will satisfy the condition. However, the choice of weights
     influences the efficiency of the estimator as illustrated in \cref{prop:eff} later.

\begin{proposition}
\label{prop:equal}
Assuming equation~\eqref{eq:esteq} and regularity condition for the asymptotic normality of
     current ML-assisted estimator holds. For any M-estimation problem,
we have
\[
n^{\frac{1}{2}}(\wh{\bt}\left(\diag(\bomega_\textnormal{ele})\C\right) - \wh{\bt}_{\textnormal
     {\texttt{PSPA}}}) \coD
\0,
n^{\frac{1}{2}}(\wh{\bt}\left(\diag(\bomega_\textnormal{tr})\C\right) - \wh{\bt}_{\textnormal
     {\texttt{PPI++}}}) \coD \0,
n^{\frac{1}{2}}(\wh{\bt}\left(\diag(\1)\C\right) - \wh{\bt}_{\textnormal{\texttt{PPI}}})\coD
\0.
\]

Here, \(\bomega_\textnormal{ele} = [\omega_{\textnormal{ele}, 1}, \dots, \omega_{\textnormal{
     ele}, K}] \trans \in \RR^K\) and \(\omega_{\textnormal{ele}, k}\) minimizing the $k$-th
     diagonal element of \(\V(\wh{\bt}(\bomega))\), \(\omega_{tr}\) is a scalar used to
     minimize the trace of \(\V(\wh{\bt}(\bomega))\), and \(\C\) is a matrix associated with
     the second derivatives of the loss function in M-estimation, with further details
     deferred to \cref{app:proof}.
\end{proposition}

This demonstrates that for M-estimation problems, our method is asymptotically equivalent to \
     texttt{PSPA}, \texttt{PPI++}, and \texttt{PPI} with the respective weights \(\diag(\
     bomega_\textnormal{ele})\C\), \(\diag(\bomega_\textnormal{tr})\C\), and  \(\diag(\1)\C\).
      Therefore, \PSPS~can be viewed as a generalization of these existing methods.

Our third result shows that the weights used in the
\cref{prop:equal} are not optimal. Instead, our choice of \(\bomega_0\) represents the optimal
      smooth combination of \((\wh{\bt}_{\calL}, \wh{\bmeta}_{\calL}, \wh{\bmeta}_{\calU})\)
     in terms of minimizing the asymptotic variance, while still preserving consistency.

\begin{proposition}
\label{prop:eff}
Suppose \(n^{1/2}(g(\wh{\bt}_{\calL}, \wh{\bmeta}_{\calL}, \wh{\bmeta}_{\calU}) - \bt^*) \coD
\mathcal{N}(0, \bSigma_g)\) and \(g\) is a smooth function, then \(\bSigma_{g_{kk}} \geq \
     bSigma_{\PSPS_{kk}}\)
\end{proposition}

Together with \cref{prop:equal}, our results demonstrate that our protocol provides a more
     efficient estimator compared to existing methods for the M-estimation problems.
     Furthermore, the applicability of our protocol is not limited to M-estimation and only
     requires summary statistics as input. It also indicates that in a setting of federated
     data analysis \citep{jordan2018communication} where individual-level data are not
     available, \PSPS~proves to be the optimal approach for combining shared summary
     statistics.

\begin{remark}
\texttt{PPI++} \citep{angelopoulos2023ppi++} employs a power-tuning scalar for variance
     reduction in ML-assisted inference. This scalar is obtained by minimizing the trace or
     possibly other scalarization of the estimator's variance-covariance matrix. However, the
     asymptotic variance of \PSPS~is always equal to or smaller than that of \texttt{PPI++},
     irrespective of the scalarization chosen by researchers. This advantage arises because \
     PSPS~utilizes a $K \times K$ power tuning matrix, $\bomega$, for variance reduction,
     where $K$ represents the dimensionality of parameters. This matrix facilitates
     information sharing across different parameter coordinates, thereby enhancing estimation
     precision. The choice of weighting matrix in \PSPS~also allows for element-wise variance
     reduction, reducing each diagonal element of the variance-covariance matrix. In contrast,
      the single scalar in \texttt{PPI++} can only target overall trace reduction or variance
     reduction of a specific element. A detailed example is provided in \cref{app:ppi++}. Only
      in one-dimensional parameter estimation tasks, such as mean estimation, \texttt{PPI++}
     and \PSPS~exhibit the same asymptotic variance.
```

```
\end{remark}
```

N.2 Canonical Code with Comments

```
import numpy as np
import pandas as pd
from scipy.stats import norm

def psps(est_lab_y, est_lab_yhat, est_unlab_yhat, Sigma, alpha=0.05):
    """
    Performs Post-Prediction Summary-statistics-based (PSPS) inference based on the paper
    "Task-Agnostic Machine-Learning-Assisted Inference" (Miao & Lu, arXiv:2405.20039v3).

    This function implements Algorithm 1 from the paper to combine estimates derived
    from labeled data (using true outcomes Y and ML-predicted outcomes Yhat) and
    unlabeled data (using ML-predicted outcomes Yhat) to produce a more efficient
    estimator for a K-dimensional parameter theta*. It leverages summary statistics
    (point estimates and their variance-covariance matrix) as input.

    Functionality and Correspondence to Paper:
    - Inputs correspond to summary statistics described in Algorithm 1, Step 2:
        - `est_lab_y`: Corresponds to $\hat{\theta}_{\mathcal{L}}$, the K-dimensional point
             estimate using observed Y in labeled data.
        - `est_lab_yhat`: Corresponds to $\hat{\eta}_{\mathcal{L}}$, the K-dimensional point
             estimate using predicted $\hat{f}$ in labeled data.
        - `est_unlab_yhat`: Corresponds to $\hat{\eta}_{\mathcal{U}}$, the K-dimensional point
              estimate using predicted $\hat{f}$ in unlabeled data.
        - `Sigma`: Corresponds to the 3K x 3K joint variance-covariance matrix of the
             estimators
                   $[\hat{\theta}_{\mathcal{L}}^T, \hat{\eta}_{\mathcal{L}}^T, \hat{\eta}_{\
                        mathcal{U}}^T]^T$. The structure assumes independence between
                   labeled and unlabeled data sets (covariance between estimators from
                        different sets is zero),
                   consistent with the asymptotic variance structure in Eq (1). The function
                        comment notes
                   `Sigma` is the direct variance, not the asymptotic variance scaled by n.
        - `alpha`: Significance level for confidence intervals (e.g., 0.05 for 95% CI).
    - Calculations implement Algorithm 1, Step 3:
        - `K`: Dimension of the parameter theta.
        - `v_est`: Extracts $Var(\hat{\theta}_{\mathcal{L}})$.
        - `r`: Extracts $Cov(\hat{\eta}_{\mathcal{L}}, \hat{\theta}_{\mathcal{L}}) = Cov(\hat
             {\theta}_{\mathcal{L}}, \hat{\eta}_{\mathcal{L}})^T$.
        - `v_eta_lab`: Extracts $Var(\hat{\eta}_{\mathcal{L}})$.
        - `v_eta_unlab`: Extracts $Var(\hat{\eta}_{\mathcal{U}})$.
        - `V`: Calculates $Var(\hat{\eta}_{\mathcal{L}}) + Var(\hat{\eta}_{\mathcal{U}})$.
        - `omega_0`: Computes the optimal weight matrix. Solves $V \omega_0^T = r$, yielding
                     $\omega_0^T = (Var(\hat{\eta}_{\mathcal{L}}) + Var(\hat{\eta}_{\mathcal{U
                          }}))^{-1} Cov(\hat{\eta}_{\mathcal{L}}, \hat{\theta}_{\mathcal{L}})$
                          .
                     This `omega_0` (as computed by `np.linalg.solve(V, r)`) is the transpose
                          of $\hat{\omega}_0$ defined in Algorithm 1 Step 3
                     ($\hat{\omega}_0 = [Var(\hat{\eta}_{\mathcal{L}}) + Var(\hat{\eta}_{\
                          mathcal{U}})]^{-1} Cov(\hat{\theta}_{\mathcal{L}}, \hat{\eta}_{\
                          mathcal{L}})$).
        - `est`: Calculates the PSPS point estimate $\hat{\theta}_{PSPS} = \hat{\theta}_{\
             mathcal{L}} + \omega_0 (\hat{\eta}_{\mathcal{U}} - \hat{\eta}_{\mathcal{L}})$,
             where `omega_0` is the variable computed in the code.
                 Substituting `omega_0` (the code's variable) = $\hat{\omega}_{0, paper}^T$,
                      this becomes
                 $\hat{\theta}_{PSPS} = \hat{\theta}_{\mathcal{L}} + \hat{\omega}_{0, paper}^T
                      (\hat{\eta}_{\mathcal{U}} - \hat{\eta}_{\mathcal{L}})$, which matches
                      the formula in Algorithm 1 Step 3.
        - `standard_errors`: Calculates the standard errors for each element of $\hat{\theta}_
             {PSPS}$.
                           It computes $\sqrt{diag(Var(\hat{\theta}_{\mathcal{L}}) - \omega_0
                                ^T r)}$, where `omega_0` is the variable computed in the code
                                .
                           This calculation corresponds to taking the square root of the
                                diagonal elements
                           of the PSPS variance matrix $\hat{Var}(\hat{\theta}_{PSPS})$ as
                                defined in Algorithm 1 Step 3 and Theorem 1.
                           (See Discrepancy section below).
        - `lower_ci`, `upper_ci`, `p_values`: Calculate Wald-type confidence intervals and two
             -sided p-values
                                                 based on the asymptotic normality of $\hat{\
                                                      theta}_{PSPS}$ established
                                                 in Theorem 1.

    Discrepancy between Code and Paper:
    - Variance Calculation Detail: The code calculates the variance reduction term as `np.dot(
        omega_0.T, r)` which corresponds to $\omega_0^T r = \hat{\omega}_{0,paper} Cov(\hat{\
        theta}_{\mathcal{L}}, \hat{\eta}_{\mathcal{L}})^T$. The variance formula in the paper
         (Algorithm 1, Step 3; Theorem 1) involves $Cov(\hat{\theta}_{\mathcal{L}},\hat{\eta}
        _{\mathcal{L}})^{T} \hat{\omega}_{0,paper}$. These two terms, $\hat{\omega}_{0,paper}
         Cov^T$ and $Cov^T \hat{\omega}_{0,paper}$, are generally equal only if K=1 (scalar
        case) or under specific symmetry conditions on the covariance matrix $Cov(\hat{\theta
        }_{\mathcal{L}}, \hat{\eta}_{\mathcal{L}})$, which may not hold for K>1. Thus, the
        calculated standard errors might differ slightly from a direct implementation of the
        paper's variance formula for multivariate parameters (K>1).

    Implementation Details Not Explicit in Paper's Algorithm 1:
    - Asymptotic Normality Assumption: The calculation of confidence intervals (`norm.ppf`)
        and p-values (`norm.sf`) explicitly relies on the assumption that the PSPS estimator
        is asymptotically normally distributed. This is justified by Theorem 1 in the paper.
    - Input Matrix Structure: The code assumes the input `Sigma` correctly represents the
        block variance-covariance matrix with zero covariance between estimators derived from
         independent labeled and unlabeled datasets.
    - Matrix Invertibility: The code implicitly assumes that the matrix `V` ($Var(\hat{\eta}_
        {\mathcal{L}}) + Var(\hat{\eta}_{\mathcal{U}})$) is invertible, which is necessary to
         solve for `omega_0`. Theorem 1 also requires the corresponding asymptotic variance
        matrix to be invertible.
    - Output Format: The function conveniently returns results in a pandas DataFrame, a common
         practice not specified in the paper's algorithm description.

    Parameters:
    - est_lab_y: A K-dimensional array of point estimates using Y in labeled data ($\hat{\
        theta}_{\mathcal{L}}$).
    - est_lab_yhat: A K-dimensional array of point estimates using Yhat in labeled data ($\hat
        {\eta}_{\mathcal{L}}$).
    - est_unlab_yhat: A K-dimensional array of point estimates using Yhat in unlabeled data ($
        \hat{\eta}_{\mathcal{U}}$).
    - Sigma: A 3K x 3K variance-covariance matrix for the stacked vector of the above three
        estimators.
    - alpha: Specifies the confidence level as 1 - alpha for confidence intervals.

    Returns:
    - A pandas DataFrame containing the PSPS summary statistics: 'Estimate' ($\hat{\theta}_{
        PSPS}$),
      'Std.Error' ($\sqrt{diag(\hat{Var}(\hat{\theta}_{PSPS}))}$), 'Lower.CI', 'Upper.CI', and
           'P.value'.
    """
    K = len(np.array(est_lab_y))
    Sigma_matrix = np.array(Sigma)
    # Extract components from the variance-covariance matrix Sigma
    # Corresponds to Var(theta_L) in paper notation
    v_est = Sigma_matrix[0:K, 0:K]
    # Corresponds to Cov(eta_L, theta_L) = Cov(theta_L, eta_L)^T
    r = Sigma_matrix[K:2*K, 0:K]
    # Corresponds to Var(eta_L)
    v_eta_lab = Sigma_matrix[K:2*K, K:2*K]
    # Corresponds to Var(eta_U)
    v_eta_unlab = Sigma_matrix[2*K:3*K, 2*K:3*K]

    # Calculate V = Var(eta_L) + Var(eta_U)
    V = v_eta_lab + v_eta_unlab

    # Calculate the optimal weight matrix (omega_0 in code = omega_0_paper^T)
    # Solves V * omega_0^T = r => omega_0^T = V^{-1} * r

    omega_0 = np.linalg.solve(V, r)
    # Calculate PSPS point estimate: est = theta_L + omega_0_paper^T * (eta_U - eta_L)
    # Here, omega_0 is KxK (omega_0_paper^T), (eta_U - eta_L) is Kx1 vector
    est = est_lab_y + np.dot(omega_0, (est_unlab_yhat - est_lab_yhat))

    # Calculate the variance of the PSPS estimate: Var(theta_PSPS) = Var(theta_L) - omega_0 *
        r
    # Note: See discrepancy discussion above regarding paper's formula Var(theta_L) - Cov(
        theta_L, eta_L)^T * omega_0_paper
    var_est_psps_matrix = v_est - np.dot(omega_0.T, r)
    # Ensure the variance is non-negative due to potential numerical issues
    # Taking diagonal before clipping to handle the matrix vs vector case correctly
    variances = np.diag(var_est_psps_matrix)
    standard_errors = np.sqrt(variances)

    # Calculate confidence intervals using Normal approximation (justified by Theorem 1)
    z_score = norm.ppf(1 - alpha / 2)
    lower_ci = est - z_score * standard_errors
    upper_ci = est + z_score * standard_errors

    # Calculate p-values for H0: theta_k = 0 using Normal approximation
    # Avoid division by zero standard error
    p_values = np.full(K, np.nan) # Initialize with NaN
    non_zero_se_indices = standard_errors > np.finfo(float).eps # Indices where SE is
         effectively non-zero
    if np.any(non_zero_se_indices):
        z_stat = np.abs(est[non_zero_se_indices] / standard_errors[non_zero_se_indices])
        p_values[non_zero_se_indices] = 2 * norm.sf(z_stat) # sf = 1 - cdf

    # Create output table
    output_table = pd.DataFrame({
        'Estimate': est,
        'Std.Error': standard_errors,
        'Lower.CI': lower_ci,
        'Upper.CI': upper_ci,
        'P.value': p_values
    })

    # Add metadata attribute to the DataFrame
    output_table.attrs['Table Name'] = 'psps summary'
    return output_table
```

N.3 Unit test code

```
class TestPSPSEquivalence(unittest.TestCase):

    def setUp(self):
        """Set up test fixtures."""
        # --- Data for K=1 (scalar parameter) ---
        self.est_lab_y_k1 = [0.5]
        self.est_lab_yhat_k1 = [0.6]
        self.est_unlab_yhat_k1 = [0.55]
        self.Sigma_k1 = [[1.0, 0.2, 0.0],
                         [0.2, 1.1, 0.0],
                         [0.0, 0.0, 1.2]]
        self.alpha_k1 = 0.05

        # --- Data for K=2 (vector parameter) ---
        self.est_lab_y_k2 = [0.5, 0.3]
        self.est_lab_yhat_k2 = [0.6, 0.25]
        self.est_unlab_yhat_k2 = [0.55, 0.28]

        v_est_k2 = [[1.0, 0.1], [0.1, 0.9]]
        cov_thL_etaL_k2 = [[0.2, 0.05], [0.03, 0.15]]
        cov_etaL_thL_k2 = [[0.2, 0.03], [0.05, 0.15]]
        v_eta_lab_k2 = [[1.1, 0.15], [0.15, 1.0]]
        v_eta_unlab_k2 = [[1.2, 0.12], [0.12, 1.1]]

        self.Sigma_k2 = [
            [v_est_k2[0][0], v_est_k2[0][1], cov_thL_etaL_k2[0][0], cov_thL_etaL_k2[0][1],
                 0.0, 0.0],
            [v_est_k2[1][0], v_est_k2[1][1], cov_thL_etaL_k2[1][0], cov_thL_etaL_k2[1][1],
                 0.0, 0.0],
            [cov_etaL_thL_k2[0][0], cov_etaL_thL_k2[0][1], v_eta_lab_k2[0][0], v_eta_lab_k2
                 [0][1], 0.0, 0.0],
            [cov_etaL_thL_k2[1][0], cov_etaL_thL_k2[1][1], v_eta_lab_k2[1][0], v_eta_lab_k2
                 [1][1], 0.0, 0.0],
            [0.0, 0.0, 0.0, 0.0, v_eta_unlab_k2[0][0], v_eta_unlab_k2[0][1]],
            [0.0, 0.0, 0.0, 0.0, v_eta_unlab_k2[1][0], v_eta_unlab_k2[1][1]],
        ]
        self.alpha_k2 = 0.05

        self.est_lab_y_k1_edge = [0.1]
        self.est_lab_yhat_k1_edge = [0.2]
        self.est_unlab_yhat_k1_edge = [0.15]
        self.Sigma_k1_edge = [[0.01, 0.5, 0.0],
                              [0.5,  1.0, 0.0],
                              [0.0,  0.0, 1.0]]
        self.alpha_k1_edge = 0.05


    def _run_and_compare(self, est_lab_y, est_lab_yhat, est_unlab_yhat, Sigma, alpha,
                         expect_nan_se=False):
        """Helper method to run both functions and compare their DataFrame outputs."""
        est_lab_y_arr = np.array(est_lab_y)
        est_lab_yhat_arr = np.array(est_lab_yhat)
        est_unlab_yhat_arr = np.array(est_unlab_yhat)
        Sigma_arr = np.array(Sigma)

        canonical_output = canonical_psps_func(
            est_lab_y_arr, est_lab_yhat_arr, est_unlab_yhat_arr,
            Sigma_arr, alpha
        )
        generated_output = generated_psps_func(
            est_lab_y_arr, est_lab_yhat_arr, est_unlab_yhat_arr,
            Sigma_arr, alpha
        )

        if expect_nan_se:
            self.assertTrue(canonical_output['Std.Error'].isnull().all(),
                            "Canonical Std.Error should be NaN for this edge case.")
            self.assertTrue(generated_output['Std.Error'].isnull().all(),
                            "Generated Std.Error should be NaN for this edge case.")

        assert_frame_equal(generated_output, canonical_output,
                           check_dtype=True,
                           rtol=1e-7,
                           atol=1e-7)

    def test_k1_equivalence(self):
        """Test functional equivalence for K=1 (scalar parameter)."""
        self._run_and_compare(
            self.est_lab_y_k1, self.est_lab_yhat_k1, self.est_unlab_yhat_k1,
            self.Sigma_k1, self.alpha_k1
        )

    def test_k2_equivalence(self):
        """Test functional equivalence for K=2 (vector parameter)."""
        self._run_and_compare(
            self.est_lab_y_k2, self.est_lab_yhat_k2, self.est_unlab_yhat_k2,
            self.Sigma_k2, self.alpha_k2
        )

    def test_k1_different_alpha(self):
        """Test functional equivalence for K=1 with a different alpha."""
        self._run_and_compare(
            self.est_lab_y_k1, self.est_lab_yhat_k1, self.est_unlab_yhat_k1,
            self.Sigma_k1, 0.1
        )

    def test_k1_edge_case_negative_variance(self):
        """Test K=1 case where Var(theta_PSPS) might be negative, leading to NaN Std.Error."""
        with np.errstate(invalid='ignore'):
            self._run_and_compare(
                self.est_lab_y_k1_edge, self.est_lab_yhat_k1_edge, self.est_unlab_yhat_k1_edge
                    ,
                self.Sigma_k1_edge, self.alpha_k1_edge,
                expect_nan_se=True
            )

    def test_statsmodels_derived_inputs_k1(self):
        """Test K=1 with inputs from statsmodels GLMs and bootstrap, reading data from CSV."""

        warnings.simplefilter('ignore', ConvergenceWarning)
        warnings.simplefilter('ignore', PerfectSeparationError) # UserWarning for
             PerfectSeparation
        warnings.simplefilter('ignore', RuntimeWarning)

        # 1. Load data from CSV files
        lab_file_path = "../../test_data/lab.csv"
        unlab_file_path = "../../test_data/unlab.csv"


        lab_df = pd.read_csv(lab_file_path)
        unlab_df = pd.read_csv(unlab_file_path)


        # 2. Fit logistic regression models and extract the 'X' coefficient
        try:
            model_lab_y = glm('Y ~ X', data=lab_df, family=sm.families.Binomial()).fit(disp=0)
            est_lab_y_scalar = model_lab_y.params.get('X', 0.0)

            model_lab_yhat = glm('Yhat ~ X', data=lab_df, family=sm.families.Binomial()).fit(
                 disp=0)
            est_lab_yhat_scalar = model_lab_yhat.params.get('X', 0.0)

            model_unlab_yhat = glm('Yhat ~ X', data=unlab_df, family=sm.families.Binomial()).
                 fit(disp=0)
            est_unlab_yhat_scalar = model_unlab_yhat.params.get('X', 0.0)
            var_unlab_yhat = model_unlab_yhat.bse.get('X', 0.1)**2
        except (np.linalg.LinAlgError, PerfectSeparationError, Exception) as e:
            # Catching a broader range of exceptions that might occur during fitting
            self.skipTest(f"GLM fitting failed for initial models: {e}. Skipping test.")
            return

        est_lab_y = [est_lab_y_scalar]
        est_lab_yhat = [est_lab_yhat_scalar]
        est_unlab_yhat = [est_unlab_yhat_scalar]

        # 3. Bootstrap for covariance calculation
        B = 50
        n_lab = len(lab_df)
        np.random.seed(123)

        est_lab_y_boot = np.full(B, np.nan)
        est_lab_yhat_boot = np.full(B, np.nan)
        successful_boot_fits = 0

        for i in range(B):
            boot_indices = np.random.choice(lab_df.index, size=n_lab, replace=True)
            boot_lab_df = lab_df.iloc[boot_indices].reset_index(drop=True)

            try:
                fit_y = glm('Y ~ X', data=boot_lab_df, family=sm.families.Binomial()).fit(disp
                    =0)
                est_lab_y_boot[i] = fit_y.params.get('X', est_lab_y_scalar)

                fit_yhat = glm('Yhat ~ X', data=boot_lab_df, family=sm.families.Binomial()).
                    fit(disp=0)
                est_lab_yhat_boot[i] = fit_yhat.params.get('X', est_lab_yhat_scalar)
                successful_boot_fits +=1
            except (np.linalg.LinAlgError, PerfectSeparationError, Exception):
                est_lab_y_boot[i] = est_lab_y_scalar
                est_lab_yhat_boot[i] = est_lab_yhat_scalar

        if successful_boot_fits < B / 2:
             print(f"\nWarning (test_statsmodels_derived_inputs_k1): Only {
                 successful_boot_fits}/{B} bootstrap fits succeeded. Sigma matrix might be
                 less reliable.")

        valid_boot_data = np.vstack([est_lab_y_boot, est_lab_yhat_boot])
        valid_boot_data = valid_boot_data[:, ~np.isnan(valid_boot_data).any(axis=0)]

        if valid_boot_data.shape[1] < 2:
            cov_matrix_lab = np.array([[0.1, 0.01],[0.01, 0.1]])
            print(f"\nWarning (test_statsmodels_derived_inputs_k1): Not enough valid bootstrap
                 samples ({valid_boot_data.shape[1]}) to compute reliable covariance. Using
                fallback.")
        else:
            cov_matrix_lab = np.cov(valid_boot_data)

        # 4. Compute the Sigma matrix
        Sigma = np.zeros((3, 3))
        if cov_matrix_lab.shape == (2,2):
            Sigma[0:2, 0:2] = cov_matrix_lab
        else:
            Sigma[0:2, 0:2] = np.array([[0.1, 0.01],[0.01, 0.1]]) # Fallback
            print("\nWarning (test_statsmodels_derived_inputs_k1): Covariance matrix from
                bootstrap was not 2x2. Using fallback.")

        Sigma[2, 2] = var_unlab_yhat

        alpha = 0.05

        # 5. Run and compare
        self._run_and_compare(
            est_lab_y, est_lab_yhat, est_unlab_yhat,
            Sigma, alpha
        )

        warnings.simplefilter('default', ConvergenceWarning)
        warnings.simplefilter('default', PerfectSeparationError)
        warnings.simplefilter('default', RuntimeWarning)


if __name__ == '__main__':
    # If running the file directly:
    # Ensure canonical_psps.py and generated_psps.py are in the Python path
    # and the CSV data files are at ./psps/test_data/
    unittest.main()
```

N.4 Code Generation Instruction

```
I have a instruction for code generation task:
**Function Name:**

psps

**Functionality Description:**

Implement the Post-Prediction Summary-statistics-based (PSPS) inference procedure detailed in
     Algorithm 1 of the paper "Task-Agnostic Machine-Learning-Assisted Inference" (Miao & Lu,
     arXiv:2405.20039v3). This function combines statistical estimates derived from labeled
     data (using both true and predicted outcomes) and unlabeled data (using predicted
     outcomes) to calculate a more statistically efficient estimate for a K-dimensional
     parameter, $\theta^*$. The function operates on summary statistics (point estimates and
     their joint variance-covariance matrix) provided as input.

**Input/Output Specification:**

* **Input:**
    * `est_lab_y` (list[float]): A list representing the K-dimensional point estimate vector
         obtained using true outcomes in the labeled dataset ($\hat{\theta}_{\mathcal{L}}$).
    * `est_lab_yhat` (list[float]): A list representing the K-dimensional point estimate
         vector obtained using ML-predicted outcomes in the labeled dataset ($\hat{\eta}_{\
         mathcal{L}}$).
    * `est_unlab_yhat` (list[float]): A list representing the K-dimensional point estimate
         vector obtained using ML-predicted outcomes in the unlabeled dataset ($\hat{\eta}_{\
         mathcal{U}}$).
    * `Sigma` (list[list[float]]): A 3K x 3K list of lists representing the joint variance-
         covariance matrix of the stacked estimators $[\hat{\theta}_{\mathcal{L}}^T, \hat{\eta
         }_{\mathcal{L}}^T, \hat{\eta}_{\mathcal{U}}^T]^T$. It assumes independence between
         labeled and unlabeled sets (zero covariance between estimators from different sets).
    * `alpha` (float): The significance level used for calculating confidence intervals (e.g.,
          0.05 for a 95% confidence interval). Defaults to 0.05.

* **Output:**
    * A `pandas.DataFrame` with K rows and the following columns:
        * `'Estimate'`: The PSPS point estimates ($\hat{\theta}_{PSPS}$).
        * `'Std.Error'`: The standard errors for each element of $\hat{\theta}_{PSPS}$.
        * `'Lower.CI'`: The lower bounds for the $(1-\alpha)$ confidence intervals.
        * `'Upper.CI'`: The upper bounds for the $(1-\alpha)$ confidence intervals.
        * `'P.value'`: The two-sided p-values for testing the null that each element of $\
             theta^*$ is zero.
```