BEYOND RETRIEVAL: COMPONENT RELATIONS AND FEEDBACK FOR FAITHFUL PAPER REPRODUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated paper reproduction aims to implement methods from research papers and reproduce their reported results. Existing systems retrieve related papers and code to supplement missing implementation details. However, accurate retrieval does not ensure faithful reproduction. Our diagnostic study shows that agents can unnecessarily alter reusable components while failing to implement required changes. Moreover, feedback on overall performance provides limited guidance for identifying and correcting these errors. We therefore propose COMPOREPRO, which guides code generation through component relations and refines implementations using component-level experimental feedback. On an extended ReproduceBench covering 40 papers across ten tasks in three domains, COMPOREPRO achieves an average performance gap of 9.26%, reducing the gap by 25.46 percentage points over the strongest baseline. The code will be publicly released upon acceptance.

1 INTRODUCTION

Paper reproduction is essential for validating research findings and supporting subsequent research (Peng, 2011; Stodden et al., 2016; Pineau et al., 2021; Henderson et al., 2018). However, it is often time-consuming and requires substantial domain expertise (Seo et al., 2026; Zhao et al., 2026; Lin et al., 2025b). The growing volume of research has motivated increasing interest in automated paper reproduction (Li et al., 2025; Hu et al., 2026; Seo et al., 2026). This task aims to enable agents to implement methods from paper descriptions and reproduce the reported experimental results (Starace et al., 2025; Zhao et al., 2026). By reducing manual effort in implementation and experimental debugging, it can lower the cost of validating findings and reusing existing methods (Gandhi et al., 2025; Lin et al., 2025b).

Recent work has introduced retrieval-augmented autonomous agents for paper reproduction (Hu et al., 2026; Chen et al., 2025; Xiang et al., 2025). Because papers often leave implementation details unspecified (Gundersen & Kjensmo, 2018; Seo et al., 2026), systems such as AutoReproduce (Zhao et al., 2026) and DeepCode (Li et al., 2025) retrieve related papers and open-source code to supplement the target paper. Using this context, the agents generate and execute code. They then use the overall performance gap between reproduced and reported results as feedback to refine the code (Zhou et al., 2025; Zhao et al., 2026).

Although related work provides useful context for reproduction, a key question remains:

Can agents correctly use the related work they retrieve?

A machine learning method typically consists of multiple components, such as loss functions, model architectures, and data processing pipelines (He et al., 2019; Bello et al., 2021). Some components are reused from prior work; others are modified or extended (Liu et al., 2022; Wu et al., 2021a). Ideally, reproduction should faithfully implement the changes proposed in the paper. It should also preserve the intended behavior of components reused from prior work. Such reproduction should satisfy two properties:

1 *Reliability.* Components modified or introduced by the target paper should be correctly implemented. For example, a newly proposed loss function should be implemented correctly and incorporated into training.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

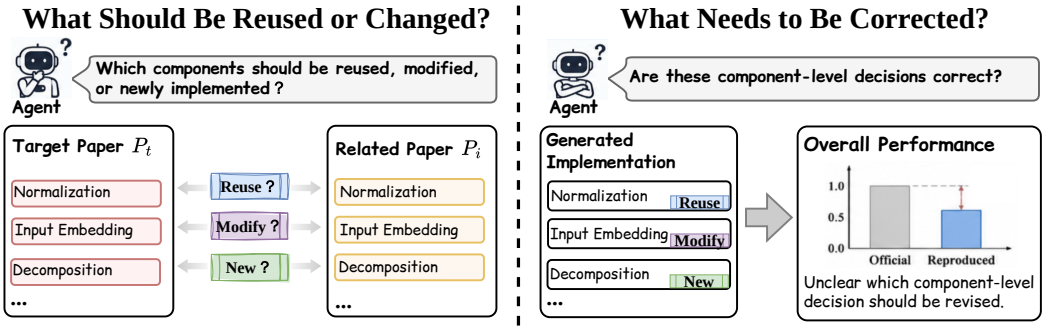


Figure 1: Two challenges in using related work for paper reproduction. **Left: What Should Be Reused or Changed?** determines which components to reuse, modify, or introduce. **Right: What Needs to Be Corrected?** checks these decisions, but overall performance alone provides limited guidance on which component requires correction.

2 *Locality.* These changes should not unintentionally alter the behavior of components reused from prior work. For example, when the paper changes only the loss function, the backbone and data processing pipeline should remain unchanged.

We conduct controlled experiments to assess the reliability and locality of state-of-the-art reproduction systems (Seo et al., 2026; Li et al., 2025; Zhao et al., 2026). Even with relevant papers and code provided, agents omit required changes and unnecessarily alter reusable components. These findings show that access to related work alone does not ensure faithful reproduction. As illustrated in Figure 1, existing agents leave component reuse and modification decisions implicit during code generation. Subsequent feedback on overall performance provides limited guidance for identifying and correcting component errors.

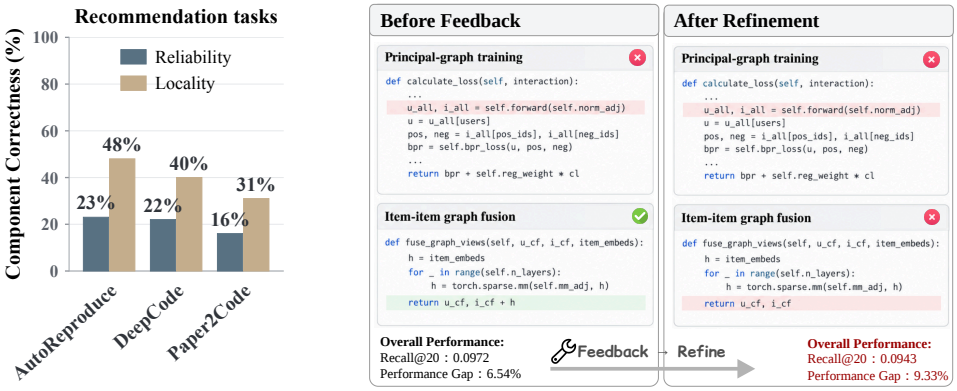
To address these limitations, we introduce CompoRepro, a component-centered framework for automated paper reproduction. CompoRepro compares the target method with related implementations using expert-defined component categories. This analysis specifies what to reuse, modify, or introduce and guides code generation. For component-level verification, we use the paper's ablation and comparison results to establish the expected effects of component changes. The agent constructs corresponding variants and measures their performance differences. Discrepancies from the reported effects then guide code inspection and refinement.

We make the following contributions:

- Our empirical analysis shows that correct retrieval alone does not ensure faithful reproduction. Agents often omit required changes or unnecessarily modify reusable components, while overall performance feedback provides limited guidance for identifying and correcting these errors.
- We introduce CompoRepro, a framework that uses component relations to guide code generation and component-level feedback to refine implementations.
- To reduce potential data contamination and broaden domain coverage, we extend ReproduceBench to include 40 papers spanning ten tasks across recommendation, time series analysis, and graph learning.
- Experiments demonstrate consistent improvements across all ten tasks. CompoRepro achieves an average Performance Gap of 9.26%, 25.46 percentage points below the strongest baseline.

2 Related Work

Recent work has explored LLM-based agents for automated paper reproduction through multi-agent collaboration and iterative code refinement (Seo et al., 2026; Lin et al., 2025a;b; Zhou et al., 2025). These agents coordinate paper understanding and code generation, while execution feedback guides subsequent revisions (Gandhi et al., 2025; Chen et al., 2025). However, target papers often leave implementation details unspecified (Semmelrock et al., 2025). Retrieval-augmented approaches



(a) Reliability and locality.

(b) PGL refinement with overall performance feedback.

Figure 2: **Diagnosing related-work utilization.** (a) Reliability and locality averaged across three recommendation tasks under oracle retrieval. (b) An illustrative PGL refinement using overall performance feedback: a graph-training error remains unresolved, while an initially correct graph-fusion component is incorrectly modified.

therefore retrieve related papers and code to supplement the available information (Zhao et al., 2026; Chen et al., 2025; Li et al., 2025; Hu et al., 2026). Retrieval follows citation links, providing context to support code generation. Our work builds on this direction by examining how agents can correctly use the retrieved context.

3 Preliminaries

3.1 Task Formulation

Given a target paper p_t, let $\mathcal{D}_{p_t}$ denote its text and $\mathcal{R}_t$ the retrieved papers and code. A reproduction agent $\mathcal{A}$ generates an executable implementation $\hat{\mathcal{C}}_{p_t} = \mathcal{A}(\mathcal{D}_{p_t}, \mathcal{R}_t)$. The objective is to faithfully implement the target method and minimize the performance gap relative to its official implementation:

$$\min_{\hat{\mathcal{C}}_{p_t}} \ \mathcal{L}\big(\mathcal{P}(\hat{\mathcal{C}}_{p_t}),\ \mathcal{P}(\mathcal{C}_{p_t})\big), \tag{1}$$

where $\mathcal{C}_{p_t}$ is the official implementation used only for evaluation, $\mathcal{P}(\cdot)$ denotes experimental performance, and $\mathcal{L}$ measures the performance discrepancy.

To compare the target method with related work, we represent each method as a set of functional components. For a domain d, experts define a shared set of component categories $\mathcal{U}^{(d)} = \{u_1, \ldots, u_{M_d}\}$, such as data processing, feature encoding, model backbones, prediction heads, loss functions, and training strategies. These components describe method-level functions rather than individual code files or classes. Given a target paper p_t and a related paper p_i, we define their relation for component $u \in \mathcal{U}^{(d)}$ as

$$r^{u}_{t,i} \in \{\textsc{Reuse},\ \textsc{Modify},\ \textsc{New}\}, \tag{2}$$

indicating that the target method directly reuses an existing component, modifies it, or introduces a component absent from p_i, respectively. A related paper may provide an implementation basis for only some target components. We therefore define these relations at the component level rather than the paper level.

3.2 Diagnosing Related-Work Utilization

We examine whether reproduction agents correctly use related work. Under *oracle retrieval*, agents receive the papers on which the target method builds and their official implementations. This setting allows us to study the use of existing implementations without retrieval errors. We evaluate three state-of-the-art reproduction agents—Paper2Code (Seo et al., 2026), DeepCode (Li et al., 2025), and AutoReproduce (Zhao et al., 2026)—on multimodal, general, and sequential recommendation.

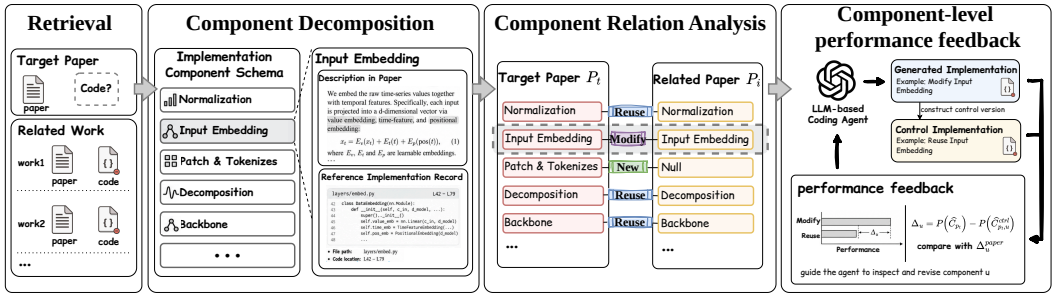


Figure 3: **Overview of COMPOREPRO.** After retrieval, expert-defined component schemas organize the target paper and related implementations. Component relation analysis identifies what to reuse, modify, or introduce and guides code generation. Component-level performance feedback compares the effects of component changes with those reported in the paper to guide iterative refinement.

For a target paper p_t, let $\mathcal{R}_t = \{(p_i, \mathcal{C}_{p_i})\}_{i=1}^{K_t}$ denote the supplied papers and implementations. The target method may reuse or modify existing components, or introduce new ones. Based on these relations to $\mathcal{R}_t$, domain experts partition its component set as

$$\mathcal{U}_t = \mathcal{U}_t^{\mathrm{R}} \dot{\cup} \mathcal{U}_t^{\mathrm{M}} \dot{\cup} \mathcal{U}_t^{\mathrm{N}}, \tag{3}$$

where the three disjoint subsets contain reused, modified, and new components, respectively. Experts also identify the corresponding implementation in $\mathcal{R}_t$ for each reused or modified component.

We assess two properties. *Reliability* measures whether modified and new components correctly implement the target design. *Locality* measures whether reused components retain the interfaces and functionality of their counterparts in related work. Three domain experts independently make binary judgments under these criteria. Let $s_t(u)$ be the fraction of experts who judge component u correct. We define

$$\mathrm{Reliability}_t = \frac{\sum_{u \in \mathcal{U}_t^{\mathrm{M}} \cup \mathcal{U}_t^{\mathrm{N}}} s_t(u)}{|\mathcal{U}_t^{\mathrm{M}}| + |\mathcal{U}_t^{\mathrm{N}}|}, \qquad \mathrm{Locality}_t = \frac{\sum_{u \in \mathcal{U}_t^{\mathrm{R}}} s_t(u)}{|\mathcal{U}_t^{\mathrm{R}}|}. \tag{4}$$

Our analysis highlights two findings.

Finding 1: Correct retrieval does not ensure reliability or locality. Even with the relevant papers and implementations provided, agents unnecessarily alter reusable components and fail to fully implement required changes. Figure 2a presents the component-level comparison. These errors suggest that accurate retrieval alone is insufficient: agents must also account for how the target method reuses and modifies components from related work.

Finding 2: Overall performance feedback cannot directly verify component effects. We next examine whether overall performance feedback can correct these errors. Figure 2b illustrates a revision of PGL (Yu et al., 2025): the agent leaves the faulty graph-training call unchanged but alters the initially correct item–item graph fusion. The required correction remains incomplete, and a reused component is disrupted. Overall performance reflects the joint effects of multiple components. Its gap alone therefore cannot establish whether modified or new components have their intended effects, or whether reused components retain their functionality.

4 THE PROPOSED METHOD

As shown in Figure 3, we propose COMPOREPRO, a three-stage framework centered on component relations to improve the reliability and locality of paper reproduction. Given a target paper, we adopt AutoReproduce's retrieval procedure (Zhao et al., 2026) to obtain related papers and code. We first identify components in the target paper and related implementations using expert-defined, task-specific categories. We then compare corresponding components to determine which to reuse or modify, as well as which new components to implement. These relations guide code generation.

Component	Representative Choices	Function
NORMALIZATION	RevIN, instance-norm, z-score	Stabilize input distribution
INPUT EMBEDDING	value embedding, time-feature, positional	Encode raw inputs into model-ready vectors
PATCH & TOKENIZE	Patch tokens, channel-mixing	Discretize and reshape the series
DECOMPOSITION	Trend/seasonal split, FFT, wavelet	Disentangle frequency components
BACKBONE	Linear, MLP, Transformer, SSM	Core sequence modeling
PREDICTION HEAD	linear projection, MLP, autoregressive	Map representations to forecasts
LOSS	MSE, MAE, quantile, Huber	Training objective
OPTIMIZATION	AdamW, OneCycle, EMA	Optimizer and learning-rate schedule

Figure 4: **Implementation Component Schema for time-series forecasting.** Each row defines one slot in the schema, with representative choices drawn from the literature (Kim et al., 2021; Nie et al., 2022; Wu et al., 2021b; Zhou et al., 2022; Vaswani et al., 2017; Gu et al., 2022; Loshchilov & Hutter, 2018; Smith & Topin, 2018).

Finally, we reproduce the paper's ablation and comparison experiments and compare the resulting performance changes with those reported. This provides component-level feedback for iterative code refinement, yielding an executable implementation of the target method.

4.1 COMPONENT DECOMPOSITION

To compare the target method with related work, we define a shared component schema for each task. Task-specific schemas account for differences in data formats and modeling procedures. Drawing on domain knowledge and representative methods (Liu et al., 2024; Boka et al., 2024; Zamanzadeh Darban et al., 2024; Jeong et al., 2026; Zhu et al., 2022), experts define

$$\mathcal{U}^{(d)} = \{u_j\}_{j=1}^{M_d}, \qquad u_j = (n_j, s_j), \tag{5}$$

where n_j and s_j denote the name and functional description of component category u_j. The same schema applies to all papers within task d. Figure 4 illustrates the schema for time-series forecasting.

Using this schema, the agent extracts component designs from the target paper's Methods and Experiments sections, including relevant descriptions and equations. For related work, it identifies components directly from the source code and records their implementation logic, file paths, and line ranges. Organizing these records by component category allows subsequent relation analysis to compare the target designs with existing implementations.

4.2 COMPONENT RELATION ANALYSIS

Components with the same functional role may differ in design. To use related implementations correctly, the agent must determine what can be reused, what requires modification, and what needs to be added. We therefore analyze these relations explicitly before code generation.

We prompt the agent to compare corresponding components in the target paper and related work and assign the relation $r^u_{t,i}$ defined in Eq. (2). For *Reuse* and *Modify*, it records the corresponding code from related work. For *Modify*, it also describes the differences between the target and existing designs. For *New*, it extracts the component's design description from the target paper and explains how the component connects to other components.

The agent considers these records across related papers before selecting an implementation. A component absent from one paper may still be available in another. The selected code and accompanying descriptions then guide generation: existing implementations provide the basis for reuse and modification, while new-component descriptions guide their implementation and integration.

4.3 COMPONENT-LEVEL PERFORMANCE FEEDBACK

Component relation analysis guides implementation choices, but the generated code may still contain errors. Overall performance reflects the combined effects of multiple components and provides limited guidance for locating these errors. We therefore use ablation and comparison results reported in the target paper to assess whether individual components have the expected effects.

Table 1: Performance Gap (%, ↓) across three domains and ten tasks. Lower values indicate better reproduction. We report the mean ± standard deviation over five independent runs.

Domain	Task	ReAct	OpenHands	Paper2Code	DeepCode	AutoReproduce	Ours
RecSys	MMRec	81.66±12.3	67.44±14.7	70.13±15.2	64.58±13.4	36.42±8.7	**18.23**±3.2
	GeneralRec	77.85±14.1	58.77±11.6	32.11±12.4	28.66±10.8	28.34±7.2	**8.41**±6.4
	SeqRec	80.33±13.5	71.09±16.2	65.88±14.8	61.24±12.1	34.76±9.1	**3.94**±0.8
Time Series	Classification	78.57±15.4	62.18±12.9	68.44±16.1	63.11±13.7	38.23±8.4	**5.74**±4.3
	LongTerm	82.11±11.8	75.33±14.3	37.66±10.5	34.22±9.2	31.87±7.6	**4.13**±2.1
	ShortTerm	76.88±16.7	55.44±13.2	64.79±14.6	60.13±12.4	35.67±8.9	**5.38**±3.3
	AnomalyDetection	80.66±13.6	73.55±15.1	67.22±13.9	62.88±11.3	40.12±9.3	**24.43**±4.1
Graph Learning	GeneralGL	79.44±14.2	60.88±11.7	66.55±15.3	61.77±12.6	33.45±8.2	**4.31**±2.0
	GSL	81.22±11.4	78.66±17.5	70.33±13.8	65.11±11.9	39.78±9.5	**13.28**±7.3
	NoisyGL	77.11±15.6	56.79±10.4	35.44±11.2	31.08±9.7	28.56±7.8	**4.76**±2.1

For a component u examined in the paper, we apply the same removal or replacement to the generated implementation $\hat{\mathcal{C}}_{p_t}$, yielding a control version $\hat{\mathcal{C}}^{\text{ctrl}}_{p_t,u}$. We train and evaluate both versions while keeping all other components and experimental conditions fixed. The resulting performance change is

$$\Delta_u = \mathcal{P}\left(\hat{\mathcal{C}}_{p_t}\right) - \mathcal{P}\left(\hat{\mathcal{C}}^{\text{ctrl}}_{p_t,u}\right). \tag{6}$$

We compare Δ_u with the corresponding change reported in the paper, Δ_u^{paper}, using the same metric and subtraction order. Discrepancies in direction or magnitude prompt the agent to inspect the relevant component code against the target design. The agent then corrects identified errors and repeats the experiments to evaluate the revision.

5 Experiments

5.1 Experimental Setup

Benchmark. We extend ReproduceBench (Zhao et al., 2026), whose original test set contains 13 papers with limited domain coverage. To broaden the evaluation and reduce potential data contamination (Jain et al., 2025), we construct a test set of 40 papers released after 2025. It covers ten tasks across recommender systems, time series analysis, and graph learning. Each paper is evaluated on two representative datasets. Benchmark details are provided in Appendix B.

Evaluation Metrics. We use *Performance Gap* from ReproduceBench (Zhao et al., 2026) to measure the discrepancy between official and reproduced results:

$$\text{Performance Gap} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\mathcal{P}(\mathcal{C}_{p_t})_i - \mathcal{P}(\hat{\mathcal{C}}_{p_t})_i|}{\max\left(\mathcal{P}(\mathcal{C}_{p_t})_i,\ \mathcal{P}(\hat{\mathcal{C}}_{p_t})_i\right)} \times 100\% \tag{7}$$

where $\mathcal{P}(\mathcal{C}_{p_t})_i$ and $\mathcal{P}(\hat{\mathcal{C}}_{p_t})_i$ denote the official and reproduced scores for evaluation i, respectively. Here, n is the number of evaluations, and a lower Performance Gap indicates better reproduction. For implementations that fail to execute, we set the reproduced scores to zero. To complement this performance-based evaluation, we also assess *method fidelity*, *experimental configuration consistency*, and *code completeness* through expert review. For each task, three domain experts independently score each criterion on a 1–10 scale (Zhao et al., 2026). We further report Fleiss' κ to quantify inter-rater agreement (Fleiss, 1971).

Baselines. We compare against general-purpose coding agents, ReAct (Yao et al., 2022) and OpenHands (Wang et al., 2025), and paper reproduction systems, Paper2Code (Seo et al., 2026), DeepCode (Li et al., 2025), and AutoReproduce (Zhao et al., 2026). AutoReproduce retrieves implementation context from related papers through paper lineage.

Table 2: Human evaluation (↑): three experts score each reproduction on three dimensions (1–10 scale).

	Method	Param.	Exp.	Avg.	κ
Paper2Code	4.67	4.23	5.12	4.67	0.71
DeepCode	5.12	4.89	5.56	5.19	0.68
AutoReproduce	6.23	5.78	6.67	6.23	0.72
Ours	**8.45**	**8.12**	**8.89**	**8.49**	0.76

Table 3: Module ablations (Performance Gap, %, ↓). RA: component relation analysis; CF: component-level feedback.

RA	CF	RecSys	Time Series
○	✔	19.38	23.46
✔	○	28.64	32.85
✔	✔	**10.19**	**9.92**

✔: retained; ○: ablated. Without CF, we use overall performance feedback.

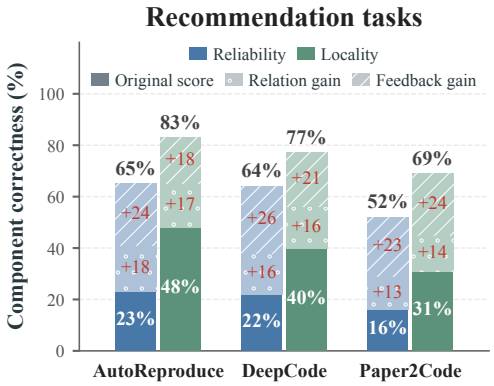


Figure 5: Reliability and locality with successive module additions. Solid segments show baseline scores; dotted and hatched segments show gains (percentage points) from component relation analysis and component-level feedback, respectively.

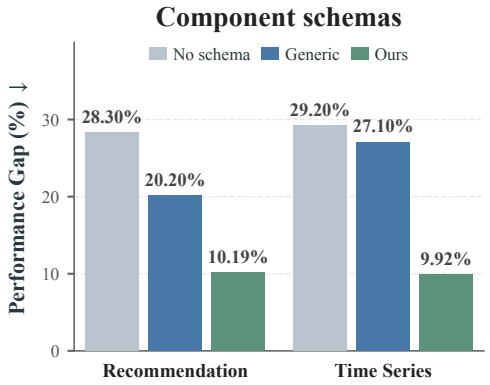


Figure 6: Performance Gap with no schema, a task-agnostic schema (Generic), and our expert-defined task-specific schema (Ours). Lower is better.

Implementation Details. All methods use Claude Opus 4.5 (Anthropic, 2025) as the backbone and run on the same hardware and software stack: 64 NVIDIA A800 80GB GPUs, PyTorch 2.7 (Paszke et al., 2019), and CUDA 12.6. We report the mean and standard deviation over five independent runs. Prompt details are provided in Appendix D.

5.2 Main Results and Ablation Study

Main Results. Table 1 shows that CompoRepro reduces the average Performance Gap from 34.72% to 9.26% compared with the strongest baseline. This brings reproduced results substantially closer to official implementations. These gains hold across all ten tasks in three domains, with lower variability across independent runs.

Human Evaluation. Table 2 reports the human evaluation results. CompoRepro achieves the highest scores on all three dimensions. The expert ratings also show substantial agreement (Fleiss' $\kappa = 0.76$) (Landis & Koch, 1977). Notably, although AutoReproduce is competitive on execution-based metrics, its average human evaluation score is lower (6.23 vs. 8.49). This gap suggests that our generated code more faithfully reflects the target paper's intended design.

Ablation Study. To assess each module's contribution, we remove component relation analysis or replace component-level performance feedback with overall performance feedback. All other settings remain fixed. Table 3 shows that both variants increase the Performance Gap, especially when component-level feedback is replaced. These results highlight the value of verifying component effects during refinement, beyond determining what to reuse or modify during generation.

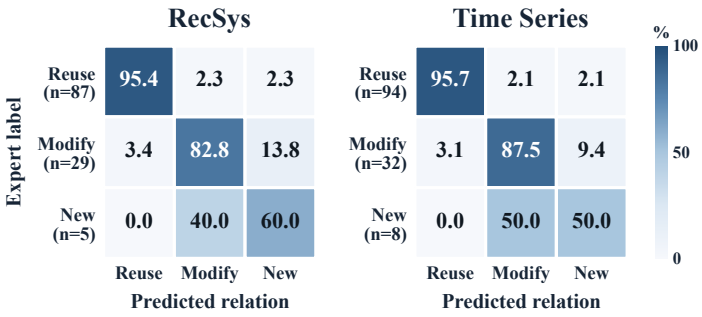


Figure 7: Row-normalized confusion matrices (%) of component relations. Rows denote expert labels and columns predictions; n gives the number of annotated components per class.

Table 4: Class-wise and macro-averaged F1 (%) from pooled component relation counts in each domain.

F1 scores (%)		
Relation	RecSys	Time Series
Reuse	97.08	97.30
Modify	84.21	84.85
New	42.86	47.06
Macro-F1	**74.71**	**76.40**

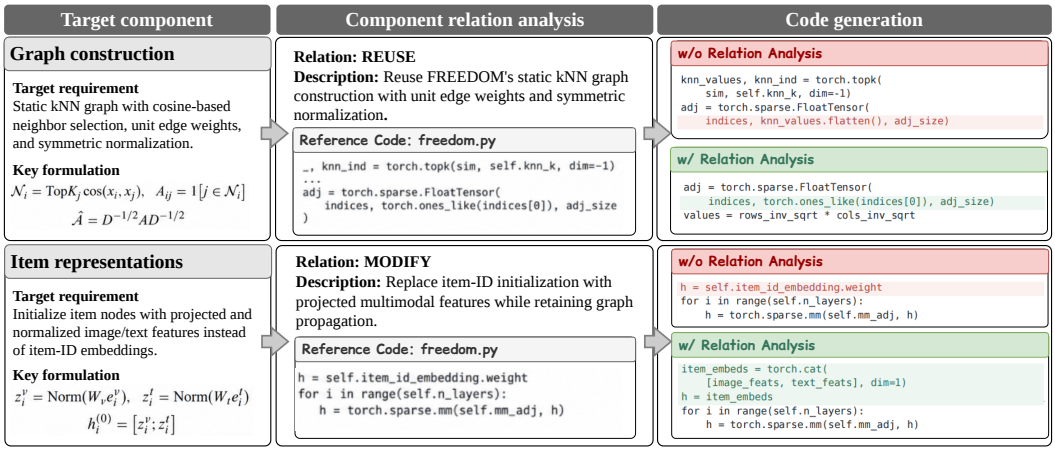


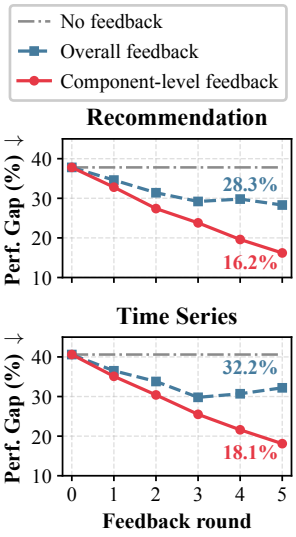
Figure 8: Component relation analysis for PGL using FREEDOM: reuse graph construction with unit edge weights and replace ID embeddings with multimodal features. Code contrasts are illustrative, not paired-run results.

Component-Level Evaluation. To assess component-level correctness, we successively add component relation analysis and component-level performance feedback to each of the three agents. As shown in Figure 5, both additions improve reliability and locality. These gains reflect more accurate implementation of required changes and better preservation of reused components.

5.3 Effectiveness of Component Relation Analysis

① **Effectiveness of Component Schemas.** We compare no schema, a task-agnostic schema, and our expert-defined task-specific schema. As shown in Figure 6, both schemas reduce the Performance Gap, with our task-specific schema achieving the lowest gaps: 10.19% for recommendation and 9.92% for time series. Its advantage over the generic schema suggests that effective decomposition should reflect the task's functional structure, providing a better basis for comparing and using related implementations.

② **Accuracy of Component Relations.** We compare predicted relations with expert annotations (Figure 7 and Table 4). Macro-F1 reaches 74.71% for recommendation and 76.40% for time series. *Reuse* and *Modify* are identified more accurately than *New*, highlighting uncertainty in distinguishing adaptation from new implementation. However, only 5 and 8 components are labeled *New* in recommendation and time series, respectively. Its F1 score is therefore sensitive to individual prediction errors.



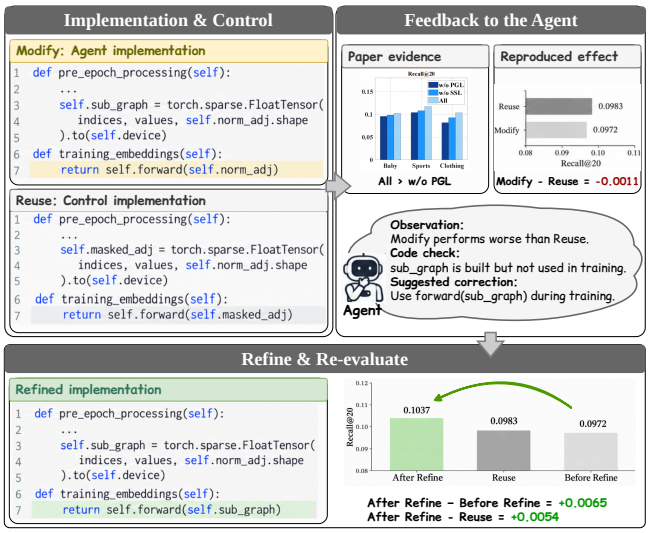


Figure 9: **Performance Gap over five feedback rounds (left; lower is better).** Round 0 denotes the shared initial implementation.

Figure 10: **Illustration of component-level feedback for PGL (right).** The agent constructs component variants and compares the resulting performance changes with those reported in PGL's ablation study. This comparison assesses whether the component has its intended effect and guides implementation refinement.

③ **Case Study.** Figure 8 illustrates how component relations guide the use of FREEDOM (Zhou & Shen, 2023) to reproduce PGL (Yu et al., 2025). Relation analysis preserves unit edge weights before normalization rather than using similarity scores as weights. It also replaces item ID embeddings with the multimodal features required by PGL while retaining graph propagation. These decisions preserve reusable designs while implementing the changes required by the target method.

5.4 Effectiveness of Component-Level Performance Feedback

① **Comparison of Feedback Strategies.** To assess how feedback guides code refinement, we compare the three strategies over five rounds from a shared initial implementation. Figure 9 reports final Performance Gaps of 16.2% and 18.1% for component-level feedback in recommendation and time series, compared with 28.3% and 32.2% for overall feedback. The latter deteriorates in time series at rounds 4 and 5. Overall performance alone therefore does not ensure continued improvement. Component-level feedback yields more consistent refinement, at approximately $1.67\times$ the execution cost.

② **Case Study.** Figure 10 illustrates component-level performance feedback for reproducing PGL. The agent modifies the graph-learning component and constructs a control following PGL's ablation setting. It compares the reproduced performance difference with that reported in the paper's ablation study. This comparison guides code inspection, revealing that training uses the full graph instead of the constructed principal subgraph. The agent then corrects the training call and re-evaluates the implementation.

6 Conclusion

We presented CompoRepro, a framework for automated paper reproduction based on component relations and component-level experimental feedback. Across ten tasks in three domains, it achieves an average Performance Gap of 9.26%. These results highlight the importance of both selecting appropriate implementations and verifying component effects.

AI USE STATEMENT

Large language models identify components using expert-defined schemas, analyze component relations, generate implementations, and refine code using experimental feedback in COMPOREPRO. Generative AI tools also assisted with language polishing. The authors reviewed all AI-assisted work and take responsibility for the manuscript, its claims, and associated artifacts.

REFERENCES

Anthropic. Claude opus 4.5 system card. Technical report, Anthropic, November 2025. URL https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf.

Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting ResNets: Improved Training and Scaling Strategies. *Advances in Neural Information Processing Systems*, 34:22614–22627, December 2021.

Tesfaye Fenta Boka, Zhendong Niu, and Rama Bastola Neupane. A survey of sequential recommendation systems: Techniques, evaluation, and future directions. *Information Systems*, 125:102427, November 2024. ISSN 0306-4379. doi: 10.1016/j.is.2024.102427.

Pengcheng Chen, Ning Yan, Zihan Zhao, Yixiao Lin, Huaibo Chen, Yue Hu, Qinbo Bai, Xiang Li, and Masood S. Mortazavi. Deep-Reproducer: From Paper Understanding to Code Generation. In *NeurIPS 2025 Fourth Workshop on Deep Learning for Code*, November 2025.

Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76 (5):378–382, 1971. ISSN 1939-1455. doi: 10.1037/h0031619.

Shubham Gandhi, Dhruv Shah, Manasi Patwardhan, Lovekesh Vig, and Gautam Shroff. ResearchCodeAgent: An LLM Multi-Agent System for Automated Codification of Research Methodologies, May 2025.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces, August 2022.

Odd Erik Gundersen and Sigbjørn Kjensmo. State of the Art: Reproducibility in Artificial Intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.11503.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of Tricks for Image Classification with Convolutional Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 558–567, 2019.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning That Matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.11694.

Xue Hu, Zewei Pan, Zhongyuan Wang, Zhou Liu, Zeli Su, and Wentao Zhang. ReproAgent: Contract-Guided Paper-to-Code Reproduction, August 2026.

Naman Jain, Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Jahoon Jeong, Youngje Oh, Donghwan Kim, and Hyunsoo Yoon. From benchmarks to interpretability: A holistic survey of deep learning for time series classification. *Knowledge-Based Systems*, 336: 115251, March 2026. ISSN 0950-7051. doi: 10.1016/j.knosys.2025.115251.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible Instance Normalization for Accurate Time-Series Forecasting against Distribution Shift. In *International Conference on Learning Representations*, October 2021.

J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, March 1977. ISSN 0006-341X.

Zongwei Li, Zhonghang Li, Zirui Guo, Xubin Ren, and Chao Huang. DeepCode: Open Agentic Coding, December 2025.

Zijie Lin, Qilin Cai, Liang Shen, and Mingjun Xiao. Enhancing Automated Paper Reproduction via Prompt-Free Collaborative Agents, December 2025a.

Zijie Lin, Yiqing Shen, Qilin Cai, He Sun, Jinrui Zhou, and Mingjun Xiao. Autop2c: An llm-based agent framework for code repository generation from multimodal content in academic papers, 2025b. URL https://arxiv.org/abs/2504.20115.

Qidong Liu, Jiaxi Hu, Yutian Xiao, Xiangyu Zhao, Jingtong Gao, Wanyu Wang, Qing Li, and Jiliang Tang. Multimodal Recommender Systems: A Survey. *ACM Computing Surveys*, 57(2):26:1–26:17, October 2024. ISSN 0360-0300. doi: 10.1145/3695461.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, September 2018.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *The Eleventh International Conference on Learning Representations*, September 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 2019.

Roger D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011. doi: 10.1126/science.1213847. URL https://www.science.org/doi/abs/10.1126/science.1213847.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché-Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program). *The Journal of Machine Learning Research*, 22(1):164:7459–164:7478, January 2021. ISSN 1532-4435.

Harald Semmelrock, Tony Ross-Hellauer, Simone Kopeinik, Dieter Theiler, Armin Haberl, Stefan Thalmann, and Dominik Kowald. Reproducibility in machine-learning-based research: Overview, barriers, and drivers. *AI Magazine*, 46(2):e70002, 2025. ISSN 2371-9621. doi: 10.1002/aaai.70002.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2Code: Automating code generation from scientific papers in machine learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 38867–38932, 2026.

Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates, May 2018.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's Ability to Replicate AI Research, April 2025.

Victoria Stodden, Marcia McNutt, David H. Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A. Heroux, John P.A. Ioannidis, and Michela Taufer. Enhancing reproducibility for computational methods. *Science*, 354(6317):1240–1241, 2016. doi: 10.1126/science.aah6168. URL https://www.science.org/doi/abs/10.1126/science.aah6168.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. *Advances in Neural Information Processing Systems*, 30, 2017.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. CvT: Introducing Convolutions to Vision Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22–31, 2021a.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, December 2021b.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. SciReplicate-Bench: Benchmarking LLMs in Agent-driven Algorithmic Reproduction from Research Papers. In *Second Conference on Language Modeling*, August 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*, September 2022.

Penghang Yu, Zhiyi Tan, Guanming Lu, and Bing-Kun Bao. Mind Individual Information! Principal Graph Learning for Multimedia Recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(12):13096–13105, April 2025. ISSN 2374-3468. doi: 10.1609/aaai.v39i12.33429.

Zahra Zamanzadeh Darban, Geoffrey I. Webb, Shirui Pan, Charu Aggarwal, and Mahsa Salehi. Deep Learning for Time Series Anomaly Detection: A Survey. *ACM Computing Surveys*, 57(1): 15:1–15:42, October 2024. ISSN 0360-0300. doi: 10.1145/3691338.

Xuanle Zhao, Zilin Sang, Yuxuan Li, Qi Shi, Weilun Zhao, Shuo Wang, Duzhen Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. AutoReproduce: Automatic AI Experiment Reproduction with Paper Lineage. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21920–21942, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1001.

Mingyang Zhou, Quanming Yao, Lun Du, Lanning Wei, and Da Zheng. Reflective Paper-to-Code Reproduction Enabled by Fine-Grained Verification, August 2025.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 27268–27286. PMLR, June 2022.

Xin Zhou and Zhiqi Shen. A Tale of Two Graphs: Freezing and Denoising Graph Structures for Multimodal Recommendation. In *Proceedings of the 31st ACM International Conference on Multimedia*, MM '23, pp. 935–943, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 979-8-4007-0108-5. doi: 10.1145/3581783.3611943.

Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. A Survey on Graph Structure Learning: Progress and Opportunities, February 2022.

A Impact Statement

This work aims to support research reproducibility by helping agents use related implementations and assess component effects against reported experiments. Generated code and reproduced results still require review: errors in component relations or experimental settings can affect fidelity, even when the code executes successfully.

B Benchmark

This appendix lists the papers used as test samples for paper reproduction, grouped by task. Paper and code links are included where available.

B.1 Recommender Systems

B.1.1 Multimodal Recommendation

Multimodal recommendation combines user–item interactions with content such as images and text. Table 5 lists the test papers for this task.

Table 5: Test papers for multimodal recommendation.

Paper title	**Date**
Harnessing Multimodal Large Language Models for Multimodal Sequential Recommendation	2025.01
NoteLLM-2: Multimodal Large Representation Models for Recommendation	2025.01
LEMUR: Large scale End-to-end MUltimodal Recommendation	2025.11
Multimodal Recommendation System Based on Cross Self-Attention Fusion	2025.01
Mind Individual Information! Principal Graph Learning for Multimedia Recommendation	2025

B.1.2 General Recommendation

General recommendation predicts user preferences from interaction histories and associated information. Table 6 lists the test papers for this task.

Table 6: Test papers for general recommendation.

Paper title	**Date**
NLGCL: Naturally Existing Neighbor Layers Graph Contrastive Learning	2025.09
FlowCF: Flow-Based Contrastive Learning for Collaborative Filtering	2025.09
CORONA: A Coarse-to-Fine Framework for Graph-based Recommendation with LLMs	2025.06
EAGER-LLM: Enhancing LLMs as Recommenders through Behavior-Semantic Integration	2025.02
Collaborative Diffusion Model for Recommender System	2025.01
CUGF: A Reliable and Fair Recommendation Framework	2025.04

B.1.3 Sequential Recommendation

Sequential recommendation models ordered interactions to predict users' subsequent choices. Table 7 lists the test papers for this task.

Table 7: Test papers for sequential recommendation.

Paper title	**Date**
Not Just What, But When: Integrating Irregular Intervals to LLM	2025.07
On-Device Large Language Models for Sequential Recommendation	2026.01
BlossomRec: Block-level Fused Sparse Attention Mechanism for Sequential Recommendations	2025.12
Dual Conditional Diffusion Models for Sequential Recommendation	2025.03
Improving Sequential Recommenders through Counterfactual Augmentation of System Exposure	2025.04

B.2 Time Series Analysis

B.2.1 Anomaly Detection

Time series anomaly detection identifies observations or intervals that deviate from expected temporal patterns. Table 8 lists the test papers for this task.

Table 8: Test papers for time series anomaly detection.

Paper title	Paper / code links	Date
TIMEMIXER++: Universal Predictive Analysis	openreview.net/pdf?id=1CLzLXSFNn github.com/kwuking/TimeMixer	2025.05
MtsCID: Capturing Coarse-Grained Dependencies	arxiv.org/pdf/2501.16364 github.com/ilwoof/MtsCID/	2025.01
CATCH: Channel-Aware Multivariate Time Series Anomaly Detection via Frequency Patching	openreview.net/pdf?id=m08aK3xxdJ github.com/decisionintelligence/CATCH	2025.01
CrossAD: Time Series Anomaly Detection with Cross-scale Associations and Cross-window Modeling	arxiv.org/pdf/2510.12489 github.com/decisionintelligence/CrossAD	2025.10

B.2.2 Classification

Time series classification assigns a category to an observed sequence. Table 9 lists the test papers for this task.

Table 9: Test papers for time series classification.

Paper title	Paper / code links	Date
TIMEMIXER++: Universal Predictive Analysis	openreview.net/pdf?id=1CLzLXSFNn github.com/kwuking/TimeMixer	2025.05
InterpretGatedNetwork: Shedding Light on TSC	openreview.net/pdf?id=n34taxF0TC github.com/YunshiWen/Ign	2025.01
SGN: Shifted Window-Based Variable Grouping	neurips.cc/virtual/2025/poster/115248 github.com/colison/SGN	2025.12

B.2.3 Long-term Forecasting

Long-term forecasting predicts future observations over extended horizons. Table 10 lists the test papers for this task.

Table 10: Test papers for long-term time series forecasting.

Paper title	Paper / code links	Date
Unlocking the Power of LSTM for Long Term	ojs.aaai.org/index.php/AAAI/article/view/33303 github.com/Eleanorkong/P-sLSTM	2025.01
TimeEmb: Lightweight Static-Dynamic	arxiv.org/abs/2510.00461 github.com/showmeon/TimeEmb	2025.10
Efficiently Enhancing Long-term Series Forecasting via Ultra-long Lookback Windows	ojs.aaai.org/index.php/AAAI/article/view/35386 github.com/SuxinTong/IRPA	2025.04
CALF: Aligning LLMs for Time Series Forecasting via Cross-modal Fine-Tuning	arxiv.org/pdf/2403.07300 github.com/Hank0626/CALF	2025.04

B.2.4 SHORT-TERM FORECASTING

Short-term forecasting predicts observations over nearer horizons using recent temporal patterns. Table 11 lists the test papers for this task.

Table 11: Test papers for short-term time series forecasting.

Paper title	Paper / code links	Date
TimeMixer++: General Time Series Pattern	openreview.net/pdf?id=1CLzLXSFNn github.com/kwuking/TimeMixer	2025.05
Chronos-2: From Univariate to Universal Forecasting	arxiv.org/pdf/2510.15821 github.com/amazon-science/chronos-forecasting	2025.10
SEMPO: Lightweight Foundation Models for Time Series Forecasting	arxiv.org/pdf/2510.19710 github.com/mala-lab/SEMPO	2025.10
Amortized Control of Continuous State Space Feynman-Kac Model for Irregular Time Series	arxiv.org/pdf/2410.05602 github.com/bw-park/ACSSM	2025.02
Selective Learning for Deep Time Series Forecasting	arxiv.org/pdf/2510.25207 github.com/GestaltCogTeam/selective-learning	2025.10

B.3 GRAPH LEARNING

B.3.1 GENERALGL

General graph learning learns representations of nodes or graphs for downstream prediction tasks such as node classification and link prediction.

B.3.2 GSL

Graph structure learning jointly refines graph topology and learns representations for downstream prediction.

B.3.3 NOISYGL

Graph learning with noisy labels evaluates prediction when node labels are unreliable or corrupted. Table 12 lists the test papers for this task.

Table 12: Test papers for graph learning with noisy labels.

Paper title	Paper link	Date
Learning from Graph: Mitigating Label Noise via Feature Reconstruction	dl.acm.org/doi/epdf/10.1145/3746252	2025.11
Training Robust Graph Neural Networks by Modeling Noise Dependencies	arxiv.org/pdf/2502.19670	2025.02
JOINT GRAPH REWIRING AND FEATURE DENOISING VIA SPECTRAL RESONANCE	openreview.net/pdf?id=zBbZ2vdLzH	2025.04
DiffGraph: Heterogeneous Graph Diffusion Model	arxiv.org/pdf/2501.02313	2025.01

C IMPLEMENTATION AND EVALUATION DETAILS

C.1 BASELINES AND SHARED SETTINGS

We evaluate two general-purpose coding agents, ReAct (Yao et al., 2022) and OpenHands (Wang et al., 2025), and three paper reproduction systems, Paper2Code (Seo et al., 2026), DeepCode (Li et al., 2025), and AutoReproduce (Zhao et al., 2026). ReAct alternates reasoning and actions with access to file operations, code execution, and web search. OpenHands uses its CodeAct-based agent.

Paper2Code organizes reproduction into planning, analysis, and code generation, while DeepCode combines implementation planning, code generation, and verification. AutoReproduce retrieves related papers and code through paper lineage. COMPOREPRO adopts this retrieval procedure and adds component relation analysis and component-level performance feedback.

All methods use Claude Opus 4.5 and the computational environment specified in Section 5.1. We report the mean and standard deviation over five independent runs. The target paper's official implementation is used for evaluation, not as input to the reproduction agent.

C.2 COMPONENT-LEVEL EVALUATION

The diagnostic experiment uses oracle retrieval: Paper2Code, DeepCode, and AutoReproduce receive the related papers on which the target method builds and their official code. Domain experts annotate the target components as *Reuse*, *Modify*, or *New*. These annotations are used for evaluation and are not supplied to the agents during code generation.

Three experts independently assess each generated component. Modified and new components must implement the target design and participate in execution. Reused components must retain the interfaces and functionality of their counterparts in related work; identical code text is not required. A component's score is the fraction of experts who judge it correct. Reliability and locality are computed as in Eq. (4), averaged over papers within each task, and then macro-averaged across the three recommendation tasks.

For relation classification, predicted labels are compared with expert annotations under the same component schema and related implementations. Confusion matrices are normalized by the number of components in each expert-labeled class. Class-wise F1 and Macro-F1 are computed from the pooled component counts within each domain. These classification scores assess relation judgments, whereas reliability and locality assess the generated implementations.

C.3 FEEDBACK COMPARISON

The feedback experiment compares no feedback, overall performance feedback, and component-level performance feedback from a shared initial implementation. Round 0 denotes this implementation, followed by five refinement rounds for the two feedback strategies. The no-feedback condition retains the round-0 implementation. The comparison fixes the number of feedback rounds, not the execution cost: component-level feedback incurs approximately $1.67\times$ the cost of overall performance feedback.

Component-level controls follow the target paper's ablation or comparison settings. Only the selected component is removed or replaced; the remaining components and experimental conditions are held fixed. The reproduced and reported performance changes use the same metric and subtraction order. A discrepancy motivates code inspection but does not by itself identify an implementation error. The agent checks the relevant code against the target design before revising it and reruns the full and control experiments to assess the change.

D PROMPT TEMPLATES

The following templates describe the three stages of COMPOREPRO in Sections 4.1–4.3. Placeholders are filled with the task-specific schema, target-paper text, related code, and execution results. Retrieval follows AutoReproduce; the templates below concern the use of the retrieved implementations.

D.1 COMPONENT DECOMPOSITION

The expert-defined schema supplies component names and functional descriptions. The agent extracts target designs from the paper and locates corresponding components directly in related code.

Component Identification

```
Task: Identify components using the supplied task-specific schema.
```

Component Identification (continued)

```
Inputs:
- Component schema: {{ component_names_and_descriptions }}
- Target Methods and Experiments: {{ target_paper_text }}
- Related repositories: {{ related_code }}

Instructions:
1. Use the same component categories for the target and related work.
2. For the target paper, extract each component's design description,
   retaining relevant passages and equations. Do not use target code.
3. For related work, inspect files, functions, and calls to identify
   each component. Record its function, implementation, and code span.
4. Do not invent missing designs or code locations. Record unresolved
   information as unknown; failure to locate code does not prove absence.

Output one record per component and source:
{
  "component": "<schema category>",
  "source": "<target paper or related repository>",
  "description": "<target design or existing implementation>",
  "paper_evidence": "<passage/equation for target; null otherwise>",
  "code_locations": ["<file:function:start-end for related code>"],
  "code": "<related component code; null for target paper>",
  "unknowns": ["<unresolved details>"]
}
```

D.2 Component Relation Analysis and Code Generation

Relation analysis compares corresponding components and retains the information needed for reuse, modification, or new implementation. Pairwise relations are considered across all retrieved papers before choosing an implementation.

Component Relation Analysis

```
Task: Compare target components with corresponding components in
related work and determine their relations.

Inputs:
- Target component records: {{ target_components }}
- Related component records and code: {{ related_components }}

Instructions:
1. For each target component and related paper, assign Reuse, Modify,
   or New according to the target design. New means the component is
   absent from that related paper, not necessarily from all candidates.
2. For Reuse, record the related code that can be used directly.
3. For Modify, record that code and describe how the target design
   differs from the existing design.
4. For New, record the target paper's design description and how the
   component connects to other components.
5. Compare candidates against the target requirements. A component
   absent from one paper may be available in another. Select a source
   for reuse or modification when suitable; implement from scratch
   only when no candidate provides a suitable basis.
6. Failure to locate code is unresolved evidence, not New. Return a
   null relation and explain the missing evidence when undecided.

Output for each target component:
{
  "component": "<schema category>",
  "pairwise_relations": [
```

Component Relation Analysis (continued)

```
    {
      "related_paper": "<source>",
      "relation": "<Reuse, Modify, or New; null if unresolved>",
      "code_locations": ["<related code spans, when available>"],
      "code": "<code for Reuse/Modify; null for New>",
      "design_difference": "<required changes for Modify>",
      "new_design": "<target description for New>",
      "connections": "<connections to other components for New>"
    }
  ],
  "selected_relation": "<Reuse, Modify, or New; null if unresolved>",
  "selected_source": "<source; null if New or unresolved>",
  "implementation_description": "<selected design and required changes>",
  "unknowns": ["<unresolved details>"]
}
```

Relation-Guided Code Generation

```
Task: Implement the target method from the component relation records.

Inputs:
- Target method and experiment settings: {{ target_paper_text }}
- Selected code and relation descriptions: {{ component_relations }}
- Runtime environment: {{ runtime_constraints }}

Instructions:
1. Follow the final selection across related papers, not a pairwise
   New label alone. Retain code selected for Reuse and preserve its
   interfaces and functionality.
2. For Modify, start from the selected code and implement the recorded
   design differences.
3. Implement New components from the target design and connect them
   to the required components. Do not leave placeholder implementations.
4. Integrate the components into the target training and evaluation
   workflow. Report assumptions where the paper leaves details open.

Return the executable implementation, its entry points, and assumptions.
```

D.3 Component-Level Performance Feedback

The agent matches the paper's component controls, compares performance changes, and inspects code before proposing a correction. Performance discrepancies guide inspection rather than establish the cause of an error.

Component-Level Feedback and Refinement

```
Task: Use component-level experiments to guide code inspection and repair.

Inputs:
- Target component designs and relations: {{ component_records }}
- Paper ablation/comparison settings and results: {{ paper_controls }}
- Current implementation: {{ current_code }}
- Full/control execution results and logs: {{ execution_results }}

Instructions:
1. For each component examined in the paper, construct the corresponding
   control by applying the same removal or replacement. Keep all other
   components and experimental conditions fixed.
2. Train and evaluate the full and control implementations. Compute
```

Component-Level Feedback and Refinement (continued)

```
   delta = full performance - control performance, using the same
   metric and subtraction order as the paper's delta_paper.
3. Compare the direction and magnitude of delta and delta_paper.
   Do not invent results for controls not reported or not executed.
4. For a discrepancy, inspect the relevant component code against the
   target design and use execution logs to check possible causes.
   A performance discrepancy alone is not proof of a code error.
5. Correct identified implementation errors. If the cause is unresolved,
   report the uncertainty instead of making an unsupported change.
6. Rerun the full implementation and corresponding control to assess
   the revision's overall performance and component-level effect.

Return a feedback record for each checked component:
{
  "component": "<component under examination>",
  "control": "<paper experiment and removal/replacement>",
  "metric": "<shared evaluation metric>",
  "delta_paper": "<reported full-minus-control change>",
  "delta_reproduced": "<measured full-minus-control change>",
  "observation": "<agreement or discrepancy>",
  "code_check": "<checked code and confirmed error, if any>",
  "correction": "<specific edit and code location, or none>",
  "reevaluation": "<full/control results after revision, if available>",
  "unknowns": ["<remaining uncertainties>"]
}
```

E LIMITATIONS

COMPOREPRO relies on task-specific component schemas defined by domain experts. Extending the framework to new tasks therefore requires additional expert input, which may limit its scalability across domains. Automating schema construction could reduce this dependence.

Component-level feedback introduces additional execution cost by training and evaluating control variants. In our feedback comparison, it incurs approximately $1.67\times$ the cost of overall performance feedback. This additional cost may limit its applicability when individual training runs are expensive.