# Failure Localization in Multi-Agent Code Generation via Knowledge-Guided and Transferable Reasoning

**Mingyang Geng[1*], Shanzhi Gu[1*], Zhipeng Liu[2], Chuanfu Xu[1], Zhaoyang Qu[1], Haotian Wang[1†]**

[1]College of Computer Science and Technology, National University of Defense Technology, Changsha, China
[2]School of Software, Northeastern University, Shenyang, China
{gengmingyang13, gushanzhi17}@nudt.edu.cn, 2310543@stu.neu.edu.cn, {xuchuanfu, qzy, wanghaotian13}@nudt.edu.cn

## Abstract

Recent advances in multi-agent Large Language Model-based code generation enable collaborative software development through role-specialized agents. However, failure localization of code generation remains challenging due to inter-agent dependencies and solution-path multiplicity. Consequently, existing prompting-based localization methods exhibit vulnerability towards semantically valid but non-canonical strategies. To address this, we propose FLKR (Failure Localization via Knowledge-guided Reasoning), an self-supervised framework that combines behavior encoding, knowledge-strategy alignment, and consistency scoring for solution-path invariant localization. To evaluate, we also introduce COFL (Code Oriented Failure Localization), the first expert-annotated benchmark for fine-grained failure localization. Experiments show FLKR outperforms state-of-the-art prompting-based baselines by up to 14 points in Fault Localization Accuracy and 45 points in Top-1 accuracy, with strong performance in divergent, real-world, and refinement-critical cases. Such results demonstrate that our proposed FLKR generalizes well to real-world software development scenarios and opens up a new direction for failure-aware refinement recommendation by providing precise and interpretable responsibility signals.

**Datasets** — https://github.com/gmy2013/FLRK

## Introduction

Recent advances in code generation have followed a clear trajectory: from early encoder-decoder models (Wang et al. 2021, 2023d,c; Chen et al. 2022) trained on parallel code-text corpora, to powerful single-agent LLMs capable of generating complete programs from natural language prompts (Jiang et al. 2024; Li et al. 2025; Zhang et al. 2024, 2023a,b). Recent researches have shifted toward multi-agent LLM systems (Wu et al. 2023; Islam, Ali, and Parvez 2024; Dong et al. 2024; Huang et al. 2023; Du et al. 2024; Bai et al. 2025; Ashrafi, Bouktif, and Mediani 2025; Cai et al. 2021; Wang et al. 2025b, 2024a; Liu et al. 2024), which distributed software development roles, such as product manager, architect, engineer, and QA engineer, across specialized LLM agents to collaboratively plan, implement, and verify complex software artifacts. This paradigm promises greater modularity, interpretability, and scalability.

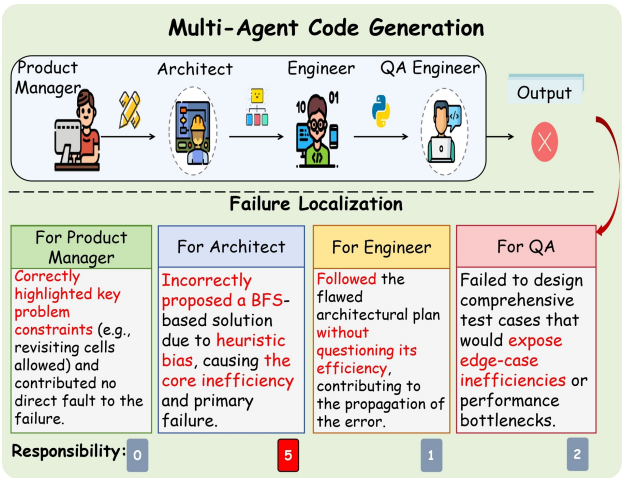


Figure 1: An illustrative example of failure localization in multi-agent code generation.

However, as coordination complexity increases, so too does the likelihood of system-level failure, often arising from miscommunication or flawed dependencies between agents. As illustrated in Figure 1, accurately localizing such failures is essential for building trustworthy and debuggable multi-agent code generation systems, yet remains an underexplored problem. Prior work on failure localization in multi-agent LLM systems (Zhang et al. 2025) primarily operates on natural language dialogue, with prompting-based techniques, e.g., full-context input, step-wise prompting, or binary search, over textual logs to query a frozen LLM for the responsible agent. Unfortunately, existing advances operate purely on surface-level text and assume strong alignment with a canonical trajectory. Consequently, the divergent structures of code data from that of textual data, together with much more accurate and qualified output requirements of the task of code generation, limit their effectiveness in code-based settings.

Therefore, the failure localization in multi-agent code generation particularly falls in the inherent solution-path

*These authors contributed equally to this work.
†Haotian Wang is the corresponding author.

multiplicity of programming tasks. On the one hand, multiple correct algorithmic strategies (e.g., greedy vs. Dynamic Programming) can yield valid implementations. On the other hand, an LLM-based system might fail due to a conceptual misunderstanding along one path, yet appear superficially different from the reference. Moreover, failures often stem from inter-agent dependencies, e.g., an upstream design flaw affecting downstream components, making it difficult to isolate responsibility. Compounding this, the absence of fine-grained supervision or labelled ground truth at the agent level further complicates learning-based approaches. As a consequence, effective localization thus requires not only structural reasoning over causal agent chains, but also semantic alignment with abstract algorithmic intent rather than surface-level code comparison.

To address the above challenges, we propose an unsupervised framework named **F**ailure **L**ocalization via **K**nowledge-guided **R**easoning (FLKR), which localizes responsible agents in multi-agent code generation without human supervision. More specifically, the proposed FLKR integrates semantic encoding, strategy alignment, and consistency-based reasoning to support solution-path invariant failure localization. By encoding each agent's behavior into a latent representation, FLKR aligns the representation with multiple algorithmic strategies from a domain knowledge base. To estimate the responsibility score for each agent, FLKR is trained via a self-supervised approach with contrastive learning (Khosla et al. 2020), enabling learning from unlabeled logs. Subsequently, by quantifying each agent's deviation from both canonical strategies and contextual behavior, FLKR achieves accurate localization of failures even under structurally different yet semantically valid solutions.

To support evaluation, we construct **C**ode **O**riented **F**ailure **L**ocalization (COFL), the first benchmark with expert-annotated fine-grained failure localizations in multi-agent code generation. In concrete, COFL contains 2520 unlabeled Codeforces cases from MetaGPT (Hong et al. 2023) for training and 142 expert-annotated cases for evaluation, covering both canonical and divergent solutions. We further exploit real-world programming cases from the SoftwareDev dataset (Hong et al. 2023) to assess generalizability. On COFL, FLKR outperforms state-of-the-art baselines by up to 14 points in Fault Localization Accuracy and 45 points in Top-1 Localization Accuracy, and remains robust on divergent-solution cases. FLKR also produces more interpretable, actionable refinement suggestions and generalizes well to real-world software debugging scenarios beyond algorithmic benchmarks. Overall, the contributions of this paper are organized as follows:

- We contribute and systematically study the novel problem of *failure localization* for multi-agent LLM-based systems in code generation. To facilitate research in this area, we present COFL, the first benchmark featuring fine-grained, expert-validated annotations that trace system-level failures back to their root causes at the agent level.
- We propose an unsupervised framework for *automated failure localization* at the agent level to pinpoint faulty agents within a collaborative workflow.
- FLKR achieves state-of-the-art performance on the COFL benchmark, outperforming prompting-based baselines by up to 14 points in Fault Localization Accuracy and 45 points in Top-1 Attribution, with notable gains in cases involving divergent yet valid solution paths, real-world software tasks, and refinement recommendation quality.

## Related Work

### Failure Localization in Collaborative Systems

(Zhang et al. 2025) pioneers automated failure localization in LLM-based multi-agent systems by introducing the Who&When dataset and proposing methods to identify the responsible agent and failure step. (Li, Naito, and Shirado 2025) further investigates reasoning failures using the Hidden Profile paradigm, analyzing how information asymmetry among agents affects collaboration and failure attribution.

### Root-Cause Analysis in Machine Learning

Root-cause analysis (RCA) has been extensively studied in traditional systems, characterized by tabular data with fixed inputs and outputs for each node. Typical approaches incorporate the Structural Causal Model (SCM) to model the interconnection across nodes, which includes discovery-based (Ikram et al. 2022) and counterfactual-based (Budhathoki et al. 2022) frameworks. To be specific, the former class, i.e., causal discovery-based RCA methods, constructs causal graphs using statistical or domain-informed methods (Glymour, Zhang, and Spirtes 2019) (e.g., PC algorithm (Ikram et al. 2022), DAG-GNN (Zheng et al. 2024)), followed by ranking (Ma et al. 2019) or search-based (Chen et al. 2014; Kim, Sumbaly, and Shah 2013) identification of root causes. By contrast, the latter class, i.e., the counterfactual-based RCA methods, instead model system anomalies via interventions on exogenous variables while preserving structural dependencies (Budhathoki et al. 2022; Okati et al. 2024; Nguyen et al. 2024).

### Root-Cause Analysis for Software Tasks

Root-cause analysis is essential in software incident management, particularly for microservice systems, where causality aids localization (Wang et al. 2022, 2023a,b, 2025a; Yang et al. 2024). Recent approaches integrate multi-modal data to address challenges like anomaly intensity (Wang et al. 2024b), link exceptions to KPIs to reduce alert fatigue (Liu et al. 2025), and combine anomaly detection with Bayesian methods for improved accuracy (Pham, Ha, and Zhang 2024a,b). Advancements also explore LLMs for diagnostic assistance (Ahmed et al. 2023) and collaborative learning for behaviour-based root cause identification (Lu et al. 2019). In microservices, methods employ causal inference and trace analysis to localize failures (Meng et al. 2020; Li et al. 2021), underscoring the growing role of advanced analytics in RCA.

However, conventional RCA paradigms are inadequate for our problem setup, given the high-dimensional generative workflows and complex interactions among multiple LLM agents. Our work addresses this gap with a fault localization framework specifically designed for multi-agent generative systems.

## Problem Formulation

**Task Goal.** Specifically, given a collaborative multi-agent LLM system that generates code for a programming problem, the localization goal is to determine which agent(s) were responsible for the failure and what proportion of the failure can be localized to each agent. In other words, the localization task judges agents by quantifying their respective scores to the failure of the code generation process.

**Input.** The input to this task consists of the following components:

**a) Programming Problem $P$:** A description includes the problem statement, input format, and the expected output.

**b) Multi-agent Collaboration** $\log L$ : A sequence of interactions between multiple agents in the LLM system, including a set of actions, decisions, code segments, and explanations generated by agents, which are sequentially executed by the system to produce the final code $C_{\text{fail}}$. The log captures all interactions between the agents and is structured as $L = \{l_1, l_2, \ldots, l_k\}$, where each $l_i$ represents an individual agent's contribution to the process.

**Output.** The output is a failure localization vector $\mathbf{y} = \{(A_i, r_i)\}_{i=1}^{|\mathcal{A}|}$, where $A_i$ represents an agent in the multi-agent system; the responsibility score $r_i \in [0, 1]$ indicates the proportion of failure assigned to agent $A_i$ for the failure of the generated code, quantifying how much the agent contributed to the failure.

## Methodology

We propose an unsupervised framework, **F**ailure **L**ocalization via **K**nowledge-guided **R**easoning (FLKR), for localizing responsible agents in multi-agent code generation without supervision. As shown in Figure 2, FLKR consists of: (1) a Semantic Encoder that embeds agent decisions; (2) a Knowledge Projection Module that aligns behaviors with canonical strategies for solution-path invariance; (3) a Consistency Scorer measuring deviation from knowledge and context; and (4) an Unsupervised Responsibility Estimator combining these signals for fine-grained localization.

### Semantic Encoder

The goal of the Semantic Encoder is to transform the raw interaction logs of each agent into structured, high-dimensional representations that can be used for downstream knowledge alignment and graph reasoning. Each decision unit $l_i$ produced by an agent is composed of two components: role, content. We encode these elements into a unified vector using a transformer-based encoder $f_{\text{enc}}$ CodeBERT. The encoded vector is given by:

$$\mathbf{z}_i = f_{\text{enc}}\ (l_i) = \ \text{Encoder}_{\ \theta}\left([\ \text{role}_{\ i};\ \text{content}_{\ i}]\right), \qquad (1)$$

where $\mathbf{z}_i \in \mathbb{R}^d$ captures the semantic behavior of agent $i$ and serves as the foundation for knowledge-guided alignment.

### Knowledge Projection Module

The Knowledge Projection Module (KPM) facilitates solution-path invariant reasoning by aligning each agent's behavior with a set of canonical strategies retrieved from a domain knowledge base. For a given problem $P$, we obtain a set of relevant algorithmic templates:

$$\mathcal{K} = \{K_1, K_2, \ldots, K_m\}, \quad K_j = (\mathcal{S}_j, \mathcal{R}_j), \qquad (2)$$

where $\mathcal{S}_j$ denotes the code skeleton and $\mathcal{R}_j$ the reasoning chain for strategy $K_j$. Both the agent representation $\mathbf{z}_i$ and the encoded knowledge entry $K_j$ are projected into a shared latent space $\mathbb{H}$ via a trainable projection function $\phi$. The projected vectors are:

$$\tilde{\mathbf{z}}_i = \phi(\mathbf{z}_i), \quad \tilde{\mathbf{k}}_j = \phi(\text{Enc}(K_j)), \qquad (3)$$

and the alignment score between agent $i$ and knowledge entry $j$ is computed via softmax-normalized dot product:

$$\alpha_{i,j} = \frac{\exp\left(\tilde{\mathbf{z}}_i^\top \tilde{\mathbf{k}}_j\right)}{\sum_{j'=1}^{m} \exp\left(\tilde{\mathbf{z}}_i^\top \tilde{\mathbf{k}}_{j'}\right)}. \qquad (4)$$

The scalar $\alpha_{i,j} \in [0, 1]$ represents the strategy compatibility score between the agent's decision and a known correct algorithmic solution, enabling localization that is agnostic to surface-level implementation details.

### Consistency Scorer

To identify agents whose decisions deviate from algorithmic knowledge or contextual coherence, we introduce a Consistency Scorer module that provides unsupervised divergence signals. Two self-supervised criteria are used. First, the knowledge divergence score is defined as the inverse of the best-aligned strategy match:

$$d_i^{\text{knowledge}} = 1 - \max_j(\alpha_{i,j}), \qquad (5)$$

which captures how far an agent's decision strays from any known valid solution path. Second, the contextual divergence score compares each agent's representation to its neighborhood context in the interaction sequence:

$$d_i^{\text{context}} = \left\| \mathbf{z}_i - \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{z}_j \right\|, \qquad (6)$$

where $\mathcal{N}_i$ denotes the set of adjacent agents (e.g., prior roles in the workflow). These are linearly combined to obtain the overall divergence signal:

$$d_i = \lambda_1 d_i^{\text{knowledge}} + \lambda_2 d_i^{\text{context}}, \qquad (7)$$

which serves as a key unsupervised indicator of potential agent-level failure.

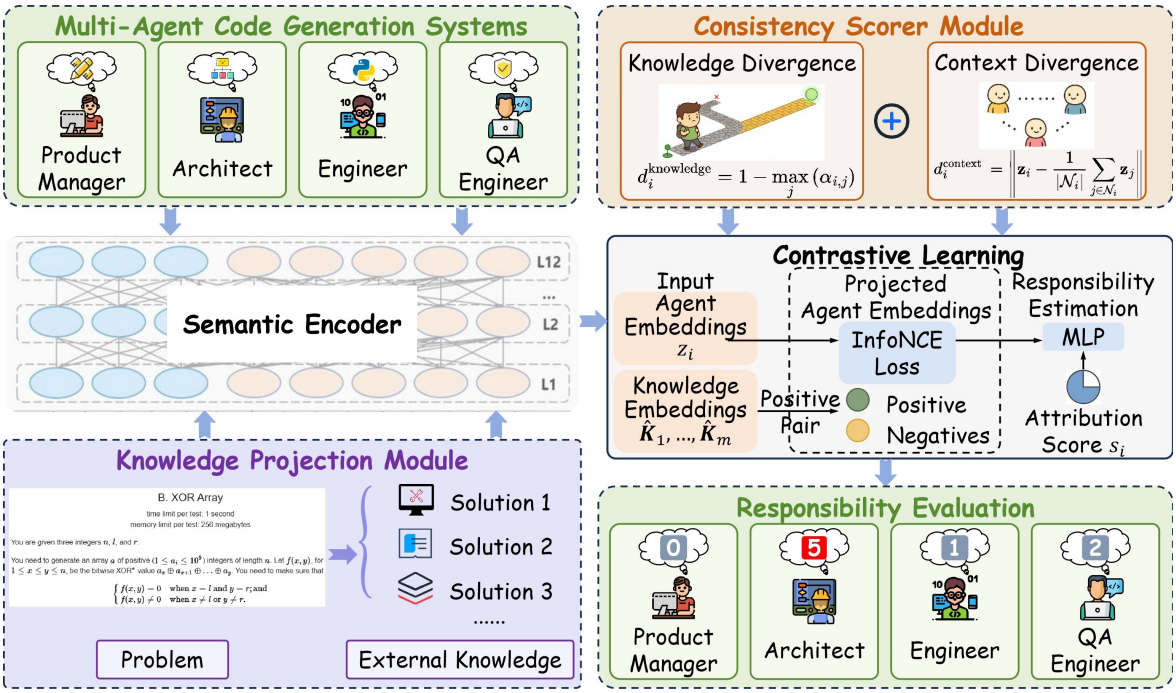


Figure 2: The architecture of our unsupervised failure localization via knowledge-guided reasoning framework.

## Unsupervised Responsibility Estimator

The final module, the Unsupervised Responsibility Estimator, combines the local consistency signal and the global causal influence of each agent to estimate a responsibility score without supervision. For each agent $i$, we fuse the semantic representation $\mathbf{z}_i$ and the divergence score $d_i$, via a lightweight scoring function:

$$s_i = \mathrm{MLP}\left([\mathbf{z}_i; d_i]\right), \tag{8}$$

where $s_i \in \mathbb{R}$ is the un-normalized localization score. These scores are then normalized across agents via a softmax operation:

$$r_i = \frac{\exp\left(s_i\right)}{\sum_j \exp\left(s_j\right)}. \tag{9}$$

The resulting vector $\mathbf{y} = \{(A_i, r_i)\}$ serves as the final agent-level failure localization, derived entirely from unsupervised signals rooted in knowledge alignment, contextual coherence, and causal graph structure.

## Training Objective and Optimization

Although FLKR is designed for unsupervised failure localization, key components of the framework are trained in a self-supervised manner using structured proxy signals derived from knowledge alignment and internal consistency. Specifically, we optimize the Knowledge Projection Module by leveraging a self-supervised similarity objective and a consistency-based pseudo-supervision signal.

The primary training loss is a contrastive knowledge alignment loss based on InfoNCE (Oord, Li, and Vinyals 2018), defined as:

$$\mathcal{L}_{\text{align}} = -\sum_{i=1}^{k} \log \frac{\exp\left(\tilde{\mathbf{z}}_i^\top \tilde{\mathbf{k}}_i^+ / \tau\right)}{\sum_{j=1}^{m} \exp\left(\tilde{\mathbf{z}}_i^\top \tilde{\mathbf{k}}_j / \tau\right)}, \tag{10}$$

where $\tilde{\mathbf{z}}_i$ is the agent projection, $\tilde{\mathbf{k}}_i^+$is the best-matching knowledge vector (positive), $\tau$ is a temperature hyperparameter, and other $\tilde{\mathbf{k}}_j$ serve as negatives.

To train the Responsibility Estimator, we add a self-distillation loss using the consistency scores $d_i$ as soft pseudo-labels:

$$\mathcal{L}_{\text{distill}} = \sum_{i=1}^{k} \left( r_i - \frac{d_i}{\sum_j d_j} \right)^2. \tag{11}$$

The overall training objective is:

$$\mathcal{L} = \mathcal{L}_{\text{align}} + \lambda \cdot \mathcal{L}_{\text{distill}}. \tag{12}$$

During training, agent representations $\mathbf{z}i$ and retrieved knowledge entries $K_j$ are projected into a shared latent space $\mathbb{H}$ via $\phi(\cdot)$. Their alignment informs both the knowledge compatibility scores $\alpha i, j$ and the self-supervised objective. The resulting consistency scores $d_i$ are used by the Responsibility Estimator. The model is trained end-to-end with external priors from the knowledge base and problem specifications.

Algorithm 1 outlines the FLKR pipeline for unsupervised agent-level failure localization. It first encodes each agent's decision into semantic vectors and projects them into a shared latent space for alignment with retrieved knowledge strategies. A consistency score is computed based on knowledge divergence and contextual deviation. Finally, the

Algorithm 1: FLKR: Unsupervised Failure Localization

**Require:** Problem $P$, agent log $L = \{l_i\}$, knowledge base $\mathcal{K}_{\text{base}}$
**Ensure:** Responsibility scores $\mathbf{y} = \{(A_i, r_i)\}$
**// Encoding and Knowledge Projection**
**for** each $l_i \in L$ **do**
  $\mathbf{z}_i \leftarrow f_{\text{enc}}(l_i), \quad \tilde{\mathbf{z}}_i \leftarrow \phi(\mathbf{z}_i)$
**end for**
$\mathcal{K} \leftarrow \text{Retrieve}(P), \quad \tilde{\mathbf{k}}_j \leftarrow \phi(\text{Enc}(K_j)) \; \forall j$
**// Consistency Scoring**
**for** each $i$ **do**
  $\alpha_{i,j} \leftarrow \text{softmax}_j(\tilde{\mathbf{z}}_i^\top \tilde{\mathbf{k}}_j)$
  $d_i \leftarrow 1 - \max_j(\alpha_{i,j}) + \lambda \cdot \|\mathbf{z}_i - \text{Mean}(\mathbf{z}_{\mathcal{N}_i})\|$
**end for**
**// Localization Estimation**
$s_i \leftarrow \text{MLP}([\mathbf{z}_i; d_i]), \quad r_i \leftarrow \frac{\exp(s_i)}{\sum_j \exp(s_j)}$
**return** $\mathbf{y} = \{(A_i, r_i)\}$

responsibility score for each agent is estimated by integrating its semantic and consistency representations, normalized into a soft localization distribution.

# Experiments

In this section, we present the evaluation part used to assess the effectiveness of our failure localization and refinement recommendation methods for multi-agent code generation systems. We define the key research questions, evaluation metrics, and experimental setup to rigorously test and benchmark the proposed approaches.

## Experimental Settings

| Component | Setting / Value |
|---|---|
| Backbone Encoder | CodeBERT (frozen) |
| Knowledge Encoder | CodeBERT (shared, frozen) |
| Projection Head $\phi(\cdot)$ | 2-layer MLP, hidden size 256 |
| Localization MLP | 2 layers, hidden size 128 |
| Optimizer | AdamW |
| Learning Rate | $1 \times 10^{-4}$ |
| Batch Size | 16 failure cases |
| Training Epochs | 20 |
| Temperature $\tau$ | 0.07 |
| Loss Weight $\lambda$ | 0.7 (distillation loss scaling) |
| Hardware | 2× NVIDIA A100 GPU (40GB) |
| Knowledge Retrieval | Top-5 strategies from curated base |

Table 1: FLKR Experimental Settings

Table 1 summarizes the implementation details and hyper-parameters used during training. The backbone encoder and knowledge encoder are based on frozen CodeBERT (Feng et al. 2020), while the projection head and localization module are trained in a self-supervised fashion. The model is trained using the AdamW optimizer, with contrastive alignment and pseudo-distillation losses. Top-5 knowledge strategies per problem are retrieved from a curated knowledge base constructed from Codeforces tags and hand-labeled algorithmic templates.

## Research Questions

The primary research questions (RQ) addressed by our evaluation are as follows:

- **RQ1:** How accurately can agent-level responsibility be localized using FLKR in multi-agent code generation?
- **RQ2:** How does solution-path invariance contribute to robust failure localization across semantically divergent but correct strategies?
- **RQ3:** Can FLKR support interpretable and actionable refinement suggestions by accurately localizing failures?
- **RQ4:** Can FLKR generalize to out-of-domain failure scenarios?
- **RQ5:** How robust is FLKR to key training variables, including training data scale, self-distillation loss weight ($\lambda$), and contrastive temperature ($\tau$)?
- **RQ6:** Can FLKR produce sharper and more focused failure localizations compared to baseline methods?

## Dataset

To support unsupervised training and evaluation, we construct a comprehensive dataset of multi-agent code generation failures. For training, we collect 2520 unlabeled failure cases from Codeforces using MetaGPT (Hong et al. 2023), each containing full agent interaction logs. This enables FLKR to learn localization signals via self-supervised objectives rooted in knowledge alignment and contextual consistency.

For evaluation, we use 142 human-annotated cases from the COFL benchmark with fine-grained responsibility vectors across diverse algorithmic domains. To assess solution-path invariance (RQ2), we include a challenging subset of 84 cases where agents pursue valid but non-canonical strategies. Additionally, to test generalizability (RQ4), we evaluate FLKR on 70 failure cases from the SoftwareDev dataset (Hong et al. 2023), covering broader software engineering tasks beyond algorithmic coding. For each case, we manually annotated the ground-truth agent-level responsibility based on failure logs and execution results.

## Evaluation Metrics

To assess the performance of our methods, we define the following evaluation metrics:

- **Fault Localization Accuracy (FLA):** Cosine similarity between predicted and ground-truth responsibility vectors, reflecting overall alignment.
- **Top-1 Localization Accuracy (Top-1 LA):** Measures whether the top-ranked predicted agent matches the true responsible agent.

| Method | General Cases | | Divergent-Solution Cases | |
|---|---|---|---|---|
| | FLA ↑ | Top-1 LA (%) ↑ | FLA ↑ | Top-1 LA (%) ↑ |
| All at once Prompting | 0.78 | 23 | 0.66 | 17 |
| Step by Step Prompting | 0.76 | 26 | 0.64 | 19 |
| Binary Search Prompting | 0.78 | 35 | 0.68 | 25 |
| w/o KPM | 0.84 | 58 | 0.72 | 43 |
| w/o context score | 0.86 | 64 | 0.74 | 48 |
| w/o knowledge score | 0.82 | 57 | 0.70 | 41 |
| w/o $\mathcal{L}_{\text{align}}$ | 0.85 | 60 | 0.73 | 46 |
| w/o $\mathcal{L}_{\text{distill}}$ | 0.87 | 70 | 0.76 | 52 |
| **FLKR** | **0.89** | **78** | **0.80** | **62** |

Table 2: Failure Localization Results on General and Divergent-Solution Cases.

## Baseline Methods

In order to evaluate the effectiveness of our proposed approach, we compare it against five baseline methods, including two ablation methods and three alternative approaches for failure localization. The following is a brief description of each baseline method:

- **All at once Prompting (Zhang et al. 2025):** The LLM receives the entire failure log in one prompt and predicts the responsible agent directly, without intermediate segmentation.
- **Step by Step Prompting (Zhang et al. 2025):** The failure log is fed sequentially, and the LLM is queried at each step to detect failures and identify the responsible agent.
- **Binary Search Prompting (Zhang et al. 2025):** The failure log is recursively split, and the LLM identifies the responsible half until the failure point is localized, balancing efficiency and granularity.
- **-Knowledge Projection (w/o KPM):** Remove the Knowledge Projection Module. Localization is based only on contextual and structural features, without leveraging domain strategies.
- **-Contextual Divergence (w/o context score):** Only use the knowledge divergence $d_i^{\text{knowledge}}$ as the unsupervised signal, ignoring surrounding agent behavior.
- **-Knowledge Divergence (w/o knowledge score):** Use only the contextual divergence $d_i^{\text{context}}$ from the neighboring agents, discarding knowledge alignment.
- **-Contrastive Learning (w/o $\mathcal{L}_{\text{align}}$):** Train the model without InfoNCE loss. The projection space is learned implicitly without strategy discrimination.
- **-Self-distillation (w/o $\mathcal{L}_{\text{distill}}$):** Remove the self-distillation loss based on consistency scores. The Responsibility Estimator learns directly from embeddings without pseudo-supervision.

## Experimental Results

**RQ1 Results: the Accuracy of FLKR** As shown in the "General Cases" column of Table 2, our proposed FLKR method achieves the highest failure localization accuracy of 0.89 and a Top-1 localization accuracy of 78%, substantially outperforming all prompting-based baselines. In particular, compared to the best-performing prompting method, Binary Search Prompting (FLA: 0.78, Top-1 LA: 35%), FLKR improves localization accuracy by 11 percentage points and more than doubles the Top-1 attribution precision. These results highlight the limitations of direct LLM prompting, which struggles to trace causal failure chains. In contrast, FLKR integrates structured reasoning over knowledge-guided alignment and consistency signals, enabling more precise localization.

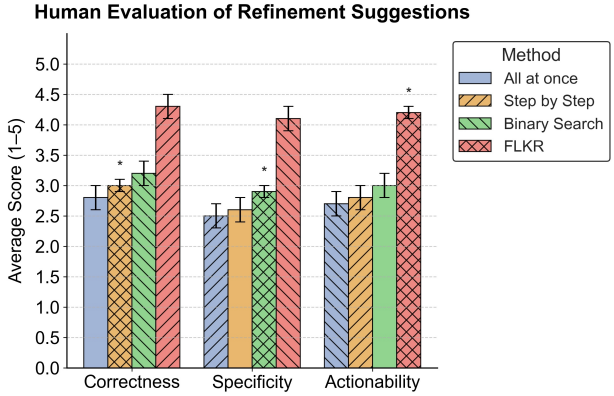


Figure 3: Human evaluation results on refinement suggestions.

**RQ2 Results: Superiority on Divergent-Solution Cases** As shown in the "Divergent-Solution Cases" column of Table 2, FLKR achieves the highest performance on divergent-solution cases, demonstrating strong solution-path invariance. Compared to prompting baselines, it localizes failure more accurately by aligning behaviors with multiple canonical strategies. Ablation results further confirm the importance of the knowledge projection module and self-distillation loss, both critical for handling ambiguity and enabling robust attribution without reliance on reference alignment.

**RQ3 Results: Human Evaluation on Supporting Interpretable and Actionable Refinements** To assess whether FLKR enables interpretable and actionable refinement, we conduct a human evaluation on 80 COFL failure cases. FLKR's localized attributions guide GPT-4 to generate targeted suggestions, which are then evaluated by annotators across three criteria. Then, the resulting code is executed on the corresponding test suite to assess whether the failure is resolved (correctness) and whether the modification specifically addresses the identified issue (specificity). Additionally, six software-experienced annotators independently score each suggestion on three dimensions: Correctness, Specificity, and Actionability, using a 1–5 Likert scale. As shown in Figure 3, FLKR outperforms prompting baselines with higher scores in Correctness (4.3), Specificity (4.1), and Actionability (4.2), confirming its effectiveness in supporting meaningful failure resolution.

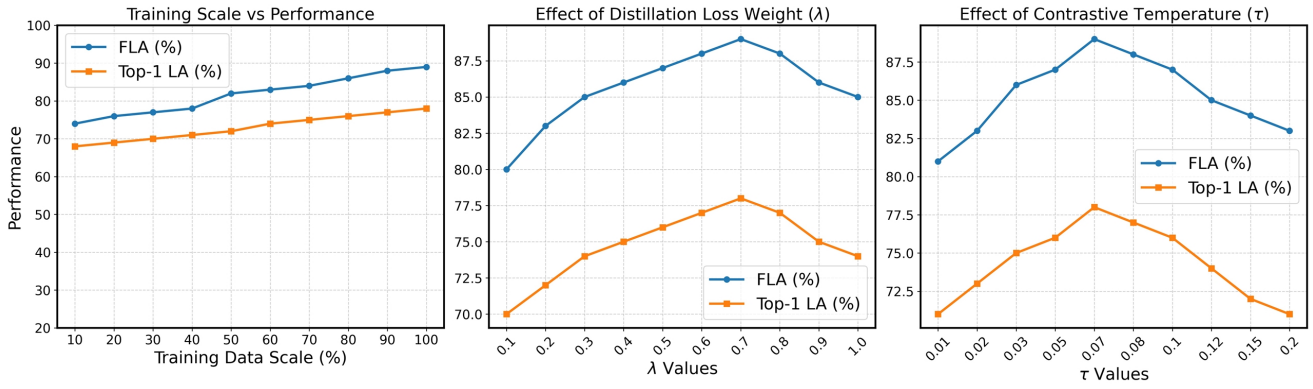


Figure 4: Sensitivity analysis to assess FLKR's robustness across key training configurations.

| Method | FLA ↑ | Top-1 LA (%) ↑ |
|---|---|---|
| All at once Prompting | 0.62 | 20 |
| Step by Step Prompting | 0.60 | 25 |
| Binary Search Prompting | 0.65 | 30 |
| **FLKR** | **0.81** | **76** |

Table 3: Generalization results on out-of-domain failure cases from the SoftwareDev dataset.

**RQ4 Results: Generalization to Out-of-Domain Failure Cases** To evaluate generalizability beyond algorithmic domains, we test FLKR and baselines on 70 manually annotated cases from the inspiration of SoftwareDev dataset (Hong et al. 2023), which includes a diverse set of real-world multi-agent software tasks. These tasks span categories such as interactive mini-games, data visualization dashboards, utility applications, algorithmic challenges, and content generators, covering a broad spectrum of software development scenarios. The predicted responsibility rankings were compared against human-labeled localizations to assess out-of-domain localization accuracy. As shown in Table 3, FLKR achieves strong out-of-domain performance (FLA 0.81, Top-1 LA 76%), while prompting-based baselines degrade under domain shift. This demonstrates FLKR's robustness in handling unfamiliar tasks through knowledge-guided, structure-aware reasoning, ensuring reliable fault localization even in scenarios outside the original training domain.

**RQ5 Results: Sensitivity Analysis on the Robustness of FLKR** FLKR's sensitivity analysis (Figure 4) shows that performance improves with increased training data and stabilizes beyond 80%, indicating data efficiency. Optimal results are achieved with a distillation weight $\lambda \approx 0.7$ and contrastive temperature $\tau$ between 0.07–0.08. These trends confirm FLKR's robustness and stable performance across key hyperparameters.

**RQ6 Results: Failure Localization Distribution Analysis** To analyze behavioral differences across attribution methods, we compute entropy over predicted responsibility vectors. A lower entropy value indicates more focused and precise attribution. On the COFL benchmark, which contains expert-annotated fine-grained failure cases, FLKR produces significantly lower entropy compared to prompting-based baselines (Figure 5). This suggests that FLKR offers sharper, more confident failure attribution, focusing its attention on specific agents responsible for the failure. The reduced entropy not only enhances accuracy but also improves the interpretability of the fault localization process, making it easier to trace back errors to the exact source, an essential feature for debugging and repair in multi-agent systems.

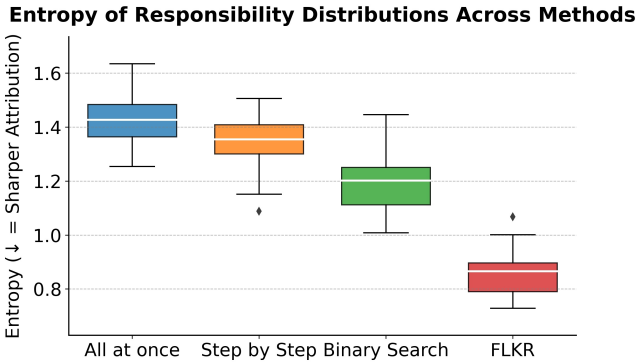


Figure 5: Entropy of responsibility distributions across methods.

## Conclusion

In this paper, we introduce FLKR, an unsupervised framework for fine-grained failure localization in multi-agent code generation. It addresses solution-path multiplicity via strategy-invariant knowledge projection and consistency reasoning. Evaluated on COFA, FLKR excels in localization accuracy, robustness to solution variance, actionable refinement, and real-world generalization, outperforming prompting baselines by up to 14% in accuracy with more interpretable refinements. Future work includes expanding benchmarks, annotating multi-agent strategies, and enhancing adaptability through interaction modeling and continual learning.

## Ethics Statement

This work involves the use of a manually annotated benchmark, where annotations were provided by trained human annotators. All annotators were informed of the purpose of the task and participated voluntarily. The annotation process adhered to ethical guidelines, ensuring privacy, confidentiality, and consistency. No sensitive personal data was involved in this study.

## Acknowledgments

This work is supported in part by the NUDT Youth Independent Innovation Science Fund under Grant No.ZK25-20.

## References

Ahmed, T.; Ghosh, S.; Bansal, C.; Zimmermann, T.; Zhang, X.; and Rajmohan, S. 2023. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 1737–1749. IEEE.

Ashrafi, N.; Bouktif, S.; and Mediani, M. 2025. Enhancing LLM Code Generation: A Systematic Evaluation of Multi-Agent Collaboration and Runtime Debugging for Improved Accuracy, Reliability, and Latency. *arXiv preprint arXiv:2505.02133*.

Bai, X.; Huang, S.; Wei, C.; and Wang, R. 2025. Collaboration between intelligent agents and large language models: A novel approach for enhancing code generation capability. *Expert Systems with Applications*, 269: 126357.

Budhathoki, K.; Minorics, L.; Blöbaum, P.; and Janzing, D. 2022. Causal structure-based root cause analysis of outliers. In *International conference on machine learning*, 2357–2369. PMLR.

Cai, H.; Zhang, X.; Lan, L.; Dong, G.; Xu, C.; Liu, X.; and Luo, Z. 2021. Learning deep discriminative embeddings via joint rescaled features and log-probability centers. *Pattern Recognition*, 114: 107852.

Chen, B.; Zhang, F.; Nguyen, A.; Zan, D.; Lin, Z.; Lou, J.-G.; and Chen, W. 2022. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*.

Chen, P.; Qi, Y.; Zheng, P.; and Hou, D. 2014. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, 1887–1895. IEEE.

Dong, Y.; Jiang, X.; Jin, Z.; and Li, G. 2024. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7): 1–38.

Du, Z.; Qian, C.; Liu, W.; Xie, Z.; Wang, Y.; Dang, Y.; Chen, W.; and Yang, C. 2024. Multi-agent software development through cross-team collaboration. *arXiv preprint arXiv:2406.08979*.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Glymour, C.; Zhang, K.; and Spirtes, P. 2019. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10: 524.

Hong, S.; Zheng, X.; Chen, J.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S. K. S.; Lin, Z.; Zhou, L.; et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4): 6.

Huang, D.; Zhang, J. M.; Luck, M.; Bu, Q.; Qing, Y.; and Cui, H. 2023. AgentCoder: Multi-Agent Code Generation with Effective Testing and Self-Optimization. *arXiv preprint arXiv:2312.13010*.

Ikram, A.; Chakraborty, S.; Mitra, S.; Saini, S.; Bagchi, S.; and Kocaoglu, M. 2022. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems*, 35: 31158–31170.

Islam, M. A.; Ali, M. E.; and Parvez, M. R. 2024. Mapcoder: Multi-agent code generation for competitive problem solving. *arXiv preprint arXiv:2405.11403*.

Jiang, X.; Dong, Y.; Wang, L.; Fang, Z.; Shang, Q.; Li, G.; Jin, Z.; and Jiao, W. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7): 1–30.

Khosla, P.; Teterwak, P.; Wang, C.; Sarna, A.; Tian, Y.; Isola, P.; Maschinot, A.; Liu, C.; and Krishnan, D. 2020. Supervised contrastive learning. *Advances in neural information processing systems*, 33: 18661–18673.

Kim, M.; Sumbaly, R.; and Shah, S. 2013. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review*, 41(1): 93–104.

Li, J.; Li, G.; Li, Y.; and Jin, Z. 2025. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology*, 34(2): 1–23.

Li, Y.; Naito, A.; and Shirado, H. 2025. Assessing Collective Reasoning in Multi-Agent LLMs via Hidden Profile Tasks. *arXiv preprint arXiv:2505.11556*.

Li, Z.; Chen, J.; Jiao, R.; Zhao, N.; Wang, Z.; Zhang, S.; Wu, Y.; Jiang, L.; Yan, L.; Wang, Z.; et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 1–10. IEEE.

Liu, J.; Yao, M.; Li, S.; Yang, D.; Wu, Z.; Qu, X.; Zhang, Z.; Li, D.; Guo, Y.; and Chen, X. 2025. Not All Exceptions Are Created Equal: Triaging Error Logs in Real World Enterprises. *ACM Transactions on Software Engineering and Methodology*.

Liu, S.; Fan, C.; Cheng, K.; Wang, Y.; Cui, P.; Sun, Y.; and Liu, Z. 2024. Inductive meta-path learning for schema-complex heterogeneous information networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Lu, Y.; Shu, Y.; Tan, X.; Liu, Y.; Zhou, M.; Chen, Q.; and Pei, D. 2019. Collaborative learning between cloud and end devices: an empirical study on location prediction. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 139–151.

Ma, M.; Lin, W.; Pan, D.; and Wang, P. 2019. Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications. In *2019 IEEE International Conference on Web Services (ICWS)*, 60–67. IEEE.

Meng, Y.; Zhang, S.; Sun, Y.; Zhang, R.; Hu, Z.; Zhang, Y.; Jia, C.; Wang, Z.; and Pei, D. 2020. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 1–10. IEEE.

Nguyen, P.; Tran, T.; Gupta, S.; Nguyen, T.; and Venkatesh, S. 2024. Root cause explanation of outliers under noisy mechanisms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20508–20515.

Okati, N.; Mejia, S. H. G.; Orchard, W. R.; Blöbaum, P.; and Janzing, D. 2024. Root Cause Analysis of Outliers with Missing Structural Knowledge. *arXiv preprint arXiv:2406.05014*.

Oord, A. v. d.; Li, Y.; and Vinyals, O. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Pham, L.; Ha, H.; and Zhang, H. 2024a. Baro: Robust root cause analysis for microservices via multivariate bayesian online change point detection. *Proceedings of the ACM on Software Engineering*, 1(FSE): 2214–2237.

Pham, L.; Ha, H.; and Zhang, H. 2024b. Root Cause Analysis for Microservice System based on Causal Inference: How Far Are We? In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 706–715.

Wang, H.; Kuang, K.; Chi, H.; Yang, L.; Geng, M.; Huang, W.; and Yang, W. 2023a. Treatment effect estimation with adjustment feature selection. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2290–2301.

Wang, H.; Kuang, K.; Lan, L.; Wang, Z.; Huang, W.; Wu, F.; and Yang, W. 2023b. Out-of-distribution generalization with causal feature separation. *IEEE Transactions on Knowledge and Data Engineering*, 36(4): 1758–1772.

Wang, H.; Li, H.; Zou, H.; Chi, H.; Lan, L.; Huang, W.; and Yang, W. 2025a. Effective and Efficient Time-Varying Counterfactual Prediction with State-Space Models. In *The Thirteenth International Conference on Learning Representations*.

Wang, H.; Yang, W.; Yang, L.; Wu, A.; Xu, L.; Ren, J.; Wu, F.; and Kuang, K. 2022. Estimating individualized causal effect with confounded instruments. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1857–1867.

Wang, M.; Li, J.; Su, H.; Yin, N.; Yang, L.; and Li, S. 2024a. GraphCL: Graph-based Clustering for Semi-Supervised Medical Image Segmentation. *arXiv preprint arXiv:2411.13147*.

Wang, M.; Ren, W.; Zhang, Y.; Fan, Y.; Shi, D.; Jing, L.; and Yin, N. 2025b. Gaussian mixture model for graph domain adaptation. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 1963–1972.

Wang, S.; Geng, M.; Lin, B.; Sun, Z.; Wen, M.; Liu, Y.; Li, L.; Bissyandé, T. F.; and Mao, X. 2023c. Natural language to code: How far are we? In *Proceedings of the 31st ACM joint European software engineering conference and symposium on the foundations of software engineering*, 375–387.

Wang, Y.; Le, H.; Gotmare, A. D.; Bui, N. D.; Li, J.; and Hoi, S. C. 2023d. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.

Wang, Y.; Wang, W.; Joty, S.; and Hoi, S. C. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.

Wang, Y.; Zhu, Z.; Fu, Q.; Ma, Y.; and He, P. 2024b. MRCA: Metric-level Root Cause Analysis for Microservices via Multi-Modal Data. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 1057–1068.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Yang, W.; Wang, H.; Li, H.; Zou, H.; Jin, R.; Kuang, K.; and Cui, P. 2024. Your neighbor matters: Towards fair decisions under networked interference. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3829–3840.

Zhang, K.; Li, J.; Li, G.; Shi, X.; and Jin, Z. 2024. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*.

Zhang, K.; Li, Z.; Li, J.; Li, G.; and Jin, Z. 2023a. Self-edit: Fault-aware code editor for code generation. *arXiv preprint arXiv:2305.04087*.

Zhang, K.; Zhang, H.; Li, G.; Li, J.; Li, Z.; and Jin, Z. 2023b. Toolcoder: Teach code generation models to use api search tools. *arXiv preprint arXiv:2305.04032*.

Zhang, S.; Yin, M.; Zhang, J.; Liu, J.; Han, Z.; Zhang, J.; Li, B.; Wang, C.; Wang, H.; Chen, Y.; et al. 2025. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*.

Zheng, L.; Chen, Z.; He, J.; and Chen, H. 2024. MULAN: multi-modal causal structure learning and root cause analysis for microservice systems. In *Proceedings of the ACM Web Conference 2024*, 4107–4116.