Fault-Controlled Routing: Proactive Failure Avoidance for Cost-Effective and Robust LLM Reasoning

1st Chunlong Fan
School of Computing
Shenyang Aerospace University
Shenyang, China
fanchl@sau.edu.cn

2nd Shanshan Chen
School of Computing
Shenyang Aerospace University
Shenyang, China
stt2083882656@qq.com

***Abstract*—Frameworks for complex reasoning tasks with Large Language Models (LLMs) are often bottlenecked by computational inefficiency, largely due to repeated exploration of invalid solution paths—a cost we term the "failure tax." To address this, we propose Fault-Controlled Routing (FCR), a framework that shifts the paradigm from reactive error correction to proactive failure avoidance. FCR combines a Risk Gate mechanism with Stratified Collaborative Representation (SCR) to enable real-time risk assessment of reasoning trajectories. Upon detecting high-recurrence failure patterns, it dynamically escalates to more robust strategies only when necessary, thereby minimizing redundant computation while maintaining strong performance. Extensive experiments on major coding benchmarks show that FCR outperforms leading baselines in accuracy and achieves substantially lower token consumption. These results demonstrate FCR's superior trade-off between performance and inference efficiency.**

***Index Terms*—LLM Agent Workflows, Failure Avoidance, Dynamic Resource Allocation, Budgeted Inference.**

I. Introduction

The paradigm of Large Language Models (LLMs) is rapidly shifting from monolithic systems to collaborative, modular frameworks, with code generation emerging as a critical application [1]. This domain not only promises to revolutionize software development productivity but also serves as an ideal testbed for model reasoning, thanks to its well-defined validation signals like unit tests [2], [3]. Consequently, an iterative "generate-verify-refine" loop has become the dominant approach [4], [5]. However, the efficiency of this cycle—specifically, how effectively it orchestrates trial-and-error to learn from failures—remains a pivotal challenge impeding its scalability.

In response, a spectrum of methodologies has emerged. These range from multi-agent orchestration frameworks like AFlow [6] and AutoGen [7] that follow pre-defined collaboration graphs, to various validation-driven workflows [8], [9], and reflective techniques such as Self-Refine [10] and Reflexion [11]. Despite their diverse strategies—spanning structural optimization to adaptive resource allocation [12]—these approaches are all fundamentally constrained by a **reactive posture toward failure**. Whether adhering to a static path or a "debug-and-repair" philosophy, they must first incur the substantial cost of a full generation and validation cycle to acquire a failure signal. This reactive model perpetuates a cycle of repeated, similar errors—a phenomenon we term the "failure tax"—and exposes the core inadequacy of after-the-fact error correction.

This shared shortcoming reveals a conspicuous research gap: the absence of a mechanism for *pre-generation failure avoidance*. Current systems lack the foresight to transform "common failure patterns" learned from historical data into low-cost, instance-level scheduling signals *before* committing to an expensive generation attempt. To bridge this gap, we introduce Fault-Controlled Routing (FCR), an intelligent, failure-aware, and cost-conscious routing framework designed to shift the paradigm from reactive correction to proactive avoidance. The core of FCR is a "Risk Gate", which assesses the risk of repeat failure after an initial low-cost attempt and proactively escalates the strategy. This intelligent routing is powered by a novel Stratified Collaborative Representation (SCR), a structured memory that decouples successful, failed, and constraint-related knowledge to enable fine-grained, evidence-based control. To validate our proactive avoidance paradigm, we conducted extensive experiments on challenging code generation benchmarks (HumanEval, MBPP, and LCB). Our results demonstrate that FCR not only achieves significant and consistent improvements in final success rates over state-of-the-art reactive methods but, more importantly, does so at a substantially lower inference cost by drastically reducing the "failure tax." These findings, which validate the efficacy and superiority of our approach, are rooted in our three primary contributions:

1) **A Proactive Routing Framework (FCR):** We introduce Fault-Controlled Routing, a framework designed to *proactively circumvent* rather than *reactively correct* recurring errors, directly addressing the inefficiencies of trial-and-error generation.
2) **An Actionable Knowledge Representation (SCR):** We propose the Stratified Collaborative Representation, a structured memory that explicitly organizes success, fail-

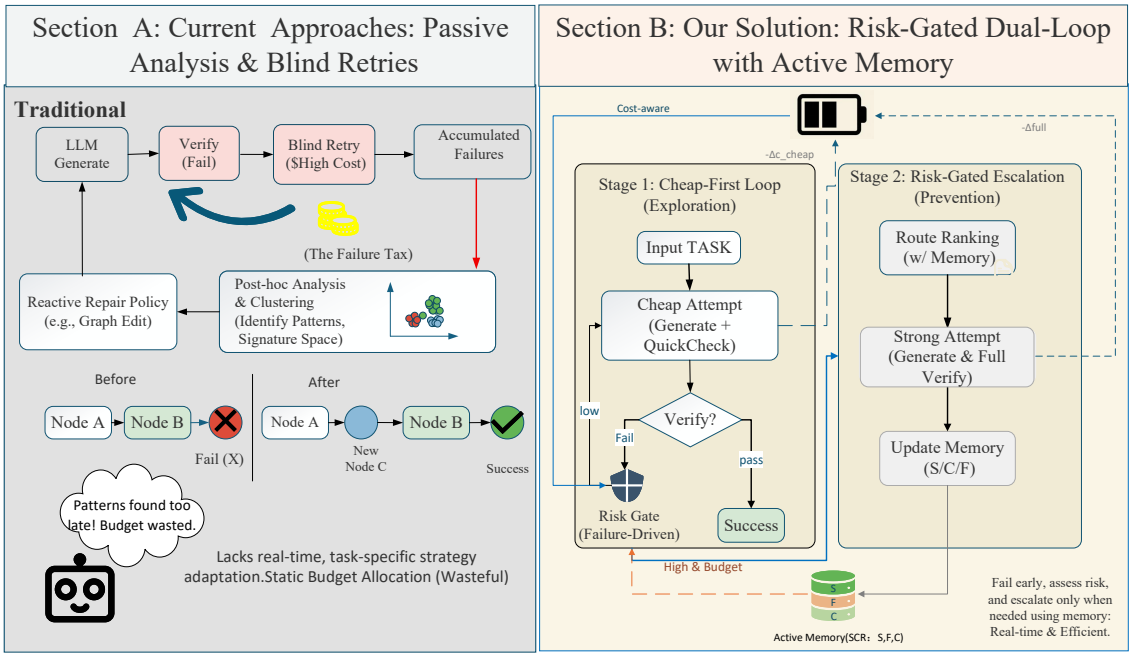

Fig. 1. An overview of our Fault-Controlled Routing (FCR) paradigm, which shifts from reactive correction to proactive failure avoidance. Left: The traditional reactive paradigm pays a high "failure tax" by repeatedly attempting similar failing paths. Right: Our FCR framework utilizes a "Risk Gate" that queries a structured knowledge base (SCR) after an initial low-cost failure. This enables the system to proactively switch to a more targeted strategy, thus minimizing redundant errors and controlling costs.

ure, and constraint knowledge. This structured separation transforms historical data into distinct, actionable signals for fine-grained process control.

3) **An Efficient, Risk-Aware Control Mechanism:** We designed and implemented a dynamic control flow centered on a "Risk Gate." This mechanism enables a resource-efficient approach by escalating to more robust strategies *only when necessary*, achieving a pragmatic balance between cost and performance.

II. Related Work

A. Workflow Representation: From Flexibility to Structure

Early workflows relied on natural language for agent coordination [4] [5], offering flexibility at the cost of formal structure. To overcome this, subsequent research introduced more structured representations, such as code-centric (e.g., AFlow [6]) and graph-based (e.g., G-Designer [13]) methods. Declarative graph representations, like MermaidFlow [14], further enhanced pre-execution consistency.

While these efforts have succeeded in defining robust static workflows [15], they universally lack a mechanism for preventative control that dynamically learns from runtime failures. FCR is designed to fill this gap; through its Stratified Collaborative Representation (SCR), it injects fine-grained, experience-driven dynamic decision-making into these static blueprints.

B. Adaptive Control: Between Reaction and Prediction

Adaptive control paradigms aim to dynamically adjust execution strategies. One dominant approach is reactive post-hoc correction, seen in methods like Self-Edit [16] and CE-Graph [9]. This is akin to "mending the fold after the sheep has escaped"; while it can improve success on a given attempt, it fails to address the fundamental problem of the "failure tax." A second approach is proactive pre-allocation, where frameworks like DAAO [12] attempt to assign resources based on a priori predictions. While forward-looking, this strategy's adaptability is limited when encountering novel failure modes, as its decisions are not grounded in real-time execution feedback. FCR creatively unifies these two directions. Its paradigm of "evidence-based proactive avoidance" marks a critical shift from post-hoc reactive correction to proactive, pre-emptive avoidance.

III. Framework

A. Problem Formulation

We formulate verification-guided code generation as a budgeted, instance-level sequential decision problem. Given a task p (with natural language description, function signature, etc.) and initial budget B (e.g., maximum tokens), the goal is to find an optimal policy π that generates code y_{final} passing all unit tests.

At each step t, π selects a route r_t based on the state x_t. The state x_t is composed of the task p, historical executions

(including the previous error e_{t-1}), and the remaining budget B_t. A route r_t specifies prompt construction, knowledge injection, and verification intensity. Executing r_t incurs cost $c(x_t, r_t)$.

The objective is:

$$\max_{\pi} P(\text{pass}|p, B, \pi) \tag{1}$$

where $P(\text{pass})$ is the probability of generating correct code within budget B.

B. Budgeted Decision Process

To implement the policy, we formalize the process over a discrete set of routes. At iteration t, the system state is $x_t = (p, e_{t-1}, B_t)$. Executing a route r_t on state x_t yields a candidate, a verification result, and incurs a cost $c(x_t, r_t)$. The budget is then updated as:

$$B_{t+1} = B_t - c(x_t, r_t) \tag{2}$$

The goal is to find a policy π that maximizes the pass probability under the budget. In practice, FCR is implemented as a heuristic budget-aware controller rather than a learned policy optimization algorithm.

C. Framework Overview

To reduce the "failure tax," we propose Fault-Controlled Routing (FCR), replacing the static generate-verify loop with a dynamic, intelligent workflow under the principle "Cheap-First, Risk-Gated Escalation" (Fig.1). This principle is realized through two synergistic modules

1) **Stratified Collaborative Representation (SCR):** A structured knowledge hub that decouples historical information into three independent stores: Success (S), Failure (F), and Constraint (C). This separation avoids signal dilution and guidance conflicts in traditional monolithic memory.
2) **Fault-Triggered Two-Stage Routing:** The execution engine that allocates resources intelligently using SCR signals. It starts with cheap attempts and escalates only when high-recurrence failure risk is detected via Risk Gate.

D. Stratified Collaborative Representation (SCR)

SCR decouples memory as $M = (S, F, C)$ to enable precise, interference-free retrieval.SCR is global within each benchmark split, and retrieval is task-conditioned.

1) Success Memory (S): stores positive examples (signature, docstring, code). Retrieval applies hard filtering on function signature followed by embedding-based soft ranking to return top-k relevant examples.

2) Failure Memory (F): captures recurrent failure patterns as clusters $f \in F$. Each cluster includes a prototype embedding, a reusable avoidance prompt, and route-specific statistics:

$$\{(n_{\text{succ}}(f, r), n_{\text{fail}}(f, r))\}_{r \in R} \tag{3}$$

FCR is agnostic to clustering method; its contribution lies in transforming clustered negatives into proactive routing signals.

3) Constraint Memory (C): provides deterministic hard rules (interface formats, security prohibitions, edge cases) as structural guardrails against low-level errors.

E. Fault-Triggered Two-Stage Routing

FCR uses cheap-first exploration followed by risk-gated escalation (Fig. 2).

1) Risk Gate: the gate probes F and computes a risk score using a Beta-LCB:

$$s_{\text{risk}}(e) = \max_{f \in \text{probe}(e)} \text{Beta-LCB}_{\gamma}(f) \tag{4}$$

Escalation to stage two occurs only if:

$$s_{\text{risk}}(e) > \tau_{\text{risk}} \tag{5}$$

This route-agnostic gating minimizes unnecessary high-cost attempts.

2) Route Ranker: Upon escalation, the ranker selects $r \in R$ by:

$$s_{\text{FR}}(e, r) = \max_{f \in \text{probe}(e)} p_{\text{succ}}(r|f) \tag{6}$$

The final score integrates SCR signals:

$$\begin{aligned} \text{Score}(r; x) = w_S \cdot \text{sim}_S + w_C \cdot |C_r| \\ + w_{\text{FR}} \cdot s_{\text{FR}}(e, r) - w_L \cdot b_L(r) \end{aligned} \tag{7}$$

where sim_S is success similarity, $|C_r|$ is satisfied constraints, s_{FR} is failure-reversal, and $b_L(r)$ is cost penalty. The highest-scoring route is executed next.

Algorithm 1 summarizes the complete FCR workflow, from cheap-first attempt to risk-driven escalation.

IV. Experiments

A. Experimental Settings

(1) **Setup:** We use three benchmarks: HumanEval [1], MBPP [2], and LiveCodeBench (LCB) [17]. Correctness is verified via unit tests. Baselines include Vanilla, CoT, PromptBreeder, ADAS, AFlow, SEW, MermaidFlow, and CE-Graph.

(2) **Implementation:** Experiments use GPT-4o-mini/Gemini-1.5-pro with consistent sampling (T=0.7, top-p=0.95). The Risk Gate's threshold, $\tau_{risk} = 0.65$, was optimized on a validation set (20% from HumanEval/MBPP) to minimize token usage with $\leq$1% accuracy loss. Performance is measured via pass@1 (correctness) and token consumption (efficiency). Baselines follow the same constraints as FCR: 300k token limit per task, 3 attempts max. Cost-optimizing baselines (e.g., SEW, CE-Graph) have their default cost controls enabled. Cheap routes use 100-token base prompts + 1-test-case verification; robust routes use 300-token prompts (with SCR knowledge injection) + full-test-suite verification, consistent across all experiments.

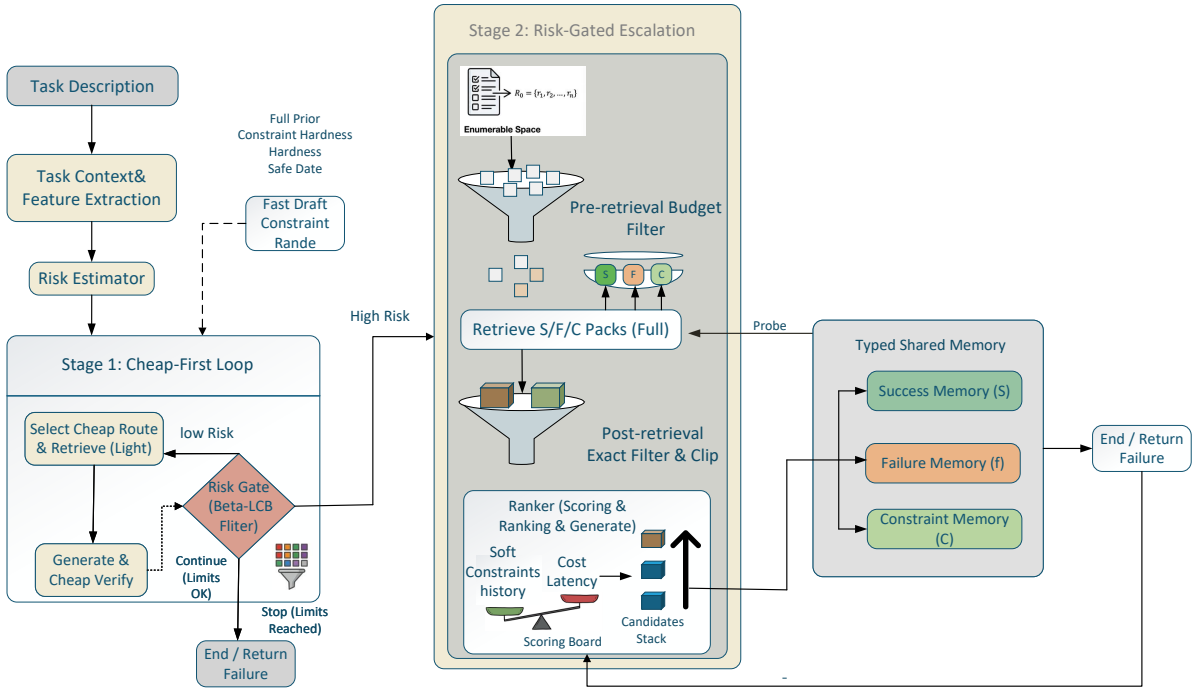

Fig. 2. The overall architecture of our Fault-Controlled Routing (FCR) framework, illustrating the "Cheap-First, Risk-Gated Escalation" workflow.

B. Main Results

As detailed in Table I, we compare FCR against several baselines across the HumanEval, MBPP, and LCB benchmarks. All evaluations were conducted on both GPT-4o-mini and Gemini-1.5-pro backbones to verify the robustness of our findings. From the results, we observe the following:

1) **On the `GPT-4o-mini` backbone, FCR demonstrates leading performance across all benchmarks.** FCR achieves `pass@1` rates of 98.47% on HumanEval, 89.44% on MBPP, and 53.25% on LCB. We attribute this advantage to FCR's ability to perform cost-aware iterative repair for each failure, whereas other baselines tend to select from a set of candidates, lacking such a fine-grained and cost-effective repair process.
2) **The performance advantage of FCR generalizes across different models.** This advantage is sustained on the `Gemini-1.5-pro` backbone, where FCR remains highly competitive on HumanEval (96.7%) and MBPP (78.1%), and outperforms SEW on LCB (49.6%). This demonstrates that FCR is a general, model-agnostic repair strategy.
3) **Risk Gating is crucial for achieving high efficiency.** Ablation studies reveal that Risk Gating is vital for maintaining cost-effectiveness. After removing this module, while the impact on accuracy is minor, the token consumption on LCB increases significantly (from 433k/1135k to 518k/1372k). This confirms that Risk Gating is a core mechanism for balancing cost and effectiveness in FCR, rather than an optional add-on.

TABLE I
COMPARISON OF VARIOUS METHODS ACROSS THREE BENCHMARKS WITH DIFFERENT BACKBONE MODELS.

Method	HumanEval	MBPP	LCB
GPT-4o-mini			
Vanilla	80.2	63.4	38.0
CoT	87.2	68.3	40.1
PromptBreeder	90.9	83.2	45.9
ADAS	88.8	73.0	42.5
AFlow	91.6	83.9	-
SEW	92.1	84.1	50.9
MermaidFlow	92.87	82.31	-
CE-Graph	94.26	88.10	-
FCR (Ours)	**98.47**	**89.44**	**53.25**
Gemini-1.5-pro			
Vanilla	79.8	61.0	36.7
CoT	86.7	68.3	39.8
PromptBreeder	88.6	71.7	44.8
SEW	89.9	74.1	47.8
FCR (Ours)	**96.7**	**78.1**	**49.6**

C. Cost-Accuracy Analysis

To evaluate the cost-accuracy trade-off, we present Pareto frontiers on HumanEval (Fig.3) and LCB (Fig.4), with costs as average dollars per problem (GPT-4o-mini pricing, 3 seeds mean $\pm$ std). In both benchmarks, the Full FCR configuration establishes a strong Pareto frontier.

On HumanEval (Fig.3), the Full configuration boosts mean accuracy to 98.47%. More critically, removing risk gating (‘noRisk‘ ablation) results in noticeably higher cost variance

Algorithm 1 Fault-Controlled Routing (FCR) Workflow

Require: Task description p, test cases T, initial budget $\mathcal{B}$, route set $\mathcal{R} = \{r_{\text{cheap}}, r_{\text{robust}_1}, \ldots, r_{\text{robust}_k}\}$
Ensure: Generated code y or failure

$S, F, C \leftarrow$ InitializeMemory() ▷ Stratified Collaborative Representation
$\mathcal{B}_t \leftarrow \mathcal{B}$ ▷ Remaining budget
$r \leftarrow r_{\text{cheap}}$ ▷ Start with the cheap route
while $\mathcal{B}_t > 0$ **do**
  $y \leftarrow$ GenerateCandidate(p, r, S, F, C)
  $\mathcal{B}_t \leftarrow \mathcal{B}_t - c(r)$ ▷ Update budget
  passed_all $\leftarrow$ true
  first_error $\leftarrow$ null
  for $t \in T$ **do**
    result, $e \leftarrow$ ExecuteAndVerify(y, t)
    if result $=$ FAILURE **then**
      passed_all $\leftarrow$ false
      first_error $\leftarrow e$
      Update$(F, (p, y, e, \text{failure}))$
      break ▷ Stop on first failure
    end if
  end for
  if passed_all **then**
    Update$(S, (p, y, \text{success}))$
    return y ▷ Success, return generated code
  end if
  $s_{\text{risk}} \leftarrow$ RiskGate(first_error, F) ▷ Beta-LCB score
  if $s_{\text{risk}} \leq \tau_{\text{risk}}$ **then**
    continue ▷ Low risk, retry with the same route
  end if
  $\mathcal{R}_{\text{robust}} \leftarrow$ FilterRoutes$(\mathcal{R}, \mathcal{B}_t)$ ▷ Escalation to Stage 2
  if $\mathcal{R}_{\text{robust}}$ is empty **then**
    break ▷ No viable robust route within budget
  end if
  $r^* \leftarrow$ RouteRanker(first_error, $S, F, C, \mathcal{R}_{\text{robust}}$)
  $r \leftarrow r^*$ ▷ Upgrade to the best robust route
end while
return failure ▷ Budget exhausted or no viable routes

and a substantially increased worst-case token upper bound (from $\tilde{1}135$k to $\tilde{1}372$k) for only a minor drop in accuracy. This proves risk gating is essential for suppressing high-risk, low-reward repairs and ensuring stable, economical convergence.

On LCB (Fig.4), this advantage is even more pronounced. The Full FCR configuration (blue) again dominates, maintaining higher accuracy with lower cost variance than the 'noRisk' variant (red) and significantly outperforming the SEW baseline (gray $\times$). This confirms the mechanism's value on more challenging, long-horizon tasks.

Overall, the Pareto analysis demonstrates that FCR's proactive risk assessment establishes a superior frontier that balances high performance with a controlled budget, significantly outperforming strong baselines in both dimensions.

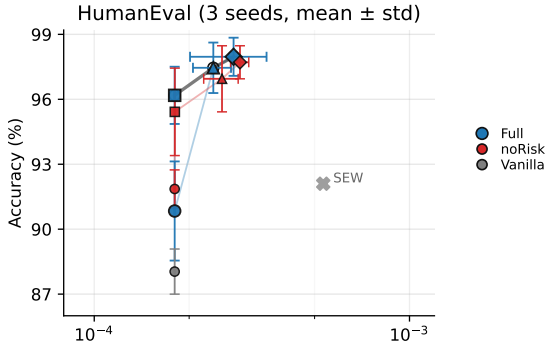

Fig. 3. **Cost-accuracy trade-off on HumanEval**. Full FCR uses $\tau_{risk} = 0.65$, the optimal value determined from the validation set.

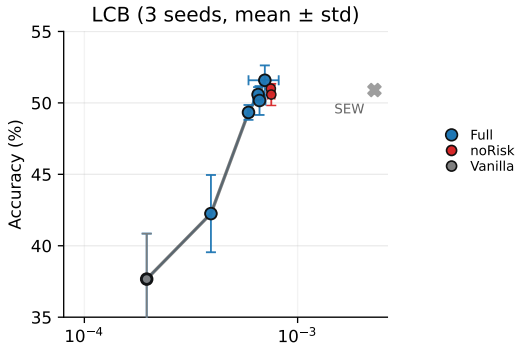

Fig. 4. **Cost-accuracy trade-off on LiveCodeBench (LCB, 3 seeds, mean ± std).** Full FCR uses $\tau_{\text{risk}} = 0.65$.

D. Ablation Studies

To dissect the contributions of each core component within the FCR framework, we conducted an ablation study. As shown in Table II, we systematically removed specific modules on the LCB and HumanEval benchmarks to observe the impact on performance (pass@1) and computational cost (Tokens).

TABLE II
ABLATION STUDY RESULTS ON LCB AND HUMANEVAL BENCHMARKS.

	LCB		HumanEval	
Method	pass@1	Tokens (k)	pass@1	Tokens (k)
FCR (Full Model)	53.25	433 ∼ 1135	98.47	61 ∼ 246
w/o Risk Gating	52.50	518 ∼ 1372	97.30	40 ∼ 163
w/o Success Memory	50.00	434 ∼ 1235	96.80	42 ∼ 170
w/o Failure Memory	51.09	340 ∼ 1261	95.42	56 ∼ 186
w/o Constraint Memory	51.50	429 ∼ 1218	97.67	58 ∼ 200
w/o Upgrade*	51.25	398 ∼ 1294	95.50	47 ∼ 190

*The alternative strategy for the "w/o Upgrade" group is to "continuously retry using the cheap route until the budget is exhausted or it succeeds."

1) **Success and Failure Memory Modules are the cornerstone of performance.** The results show that memory modules have a decisive impact, but their relative importance varies with dataset characteristics. On the more challenging and diverse LCB, removing Success Memory leads to the largest drop in `pass@1` (53.25% $\rightarrow$ 50.00%), confirming the critical role of reusing verified repair experiences across varied and complex tasks. On the relatively easier and more saturated HumanEval, removing Failure Memory causes the most significant performance decline (–3.05%), indicating that avoiding repeated known failure patterns is crucial for pushing accuracy close to saturation.
2) **Risk Gating is the core mechanism for optimizing cost-effectiveness.** Notably, on simple-task-dominated HumanEval, removing the Risk Gate cuts token consumption—eliminating its assessment overhead and unnecessary conservative upgrades for simple tasks. However, this highlights its core value in complex tasks: on LCB, it lowers token consumption upper bound from 1372k to 1135k *and* boosts accuracy (52.50%$\rightarrow$53.25%), proving its cost-optimization in critical scenarios.
3) **Constraint Memory ensures foundational repair reliability, while Strategy Upgrading is essential for breaking performance ceilings.** Disabling the former degrades performance across all benchmarks, whereas removing the latter incurs a 2.97% loss on HumanEval, proving its necessity for escaping performance plateaus even on near-saturated datasets.

Together, as a "safety guardrail" and "performance ceiling," they collectively guarantee the robustness of FCR.We additionally tested $\tau_{risk} \in \{0.5, 0.6, 0.7, 0.8\}$: FCR achieves the optimal cost-accuracy trade-off when $\tau_{risk} \in [0.6, 0.7]$ (accuracy fluctuation $\leq 0.5\%$, token consumption difference $\leq 5\%$), confirming the robustness of the Risk Gate.

E. Case Study: From Blind Retry to Targeted Repair

A case study on LCB ("Sum in a Matrix") shows baselines suffer from blind retries and high cost. FCR detects the ValueError, identifies a contract violation via the Risk Gate, and fixes it in 3 attempts with a minimal change (max(row) if row else 0). It uses 240k tokens vs. 480k for SEW.

V. Conclusion

In this work, we introduced the Fault-Controlled Routing (FCR) framework to dismantle the pervasive "failure tax" in validation-driven code generation. The core tenet of FCR is to transcend the reactive "generate-fail-reflect" cycle by repurposing historical failures into proactive, preventative signals. By constructing a Structured Collaborative Representation (SCR) of memory, FCR furnishes the language model with priori knowledge and execution guardrails. Our work culminates in a definitive conclusion: risk-aware, fine-grained scheduling is demonstrably more cost-effective than the brute-force strategy of increasing sampling depth. Future efforts will focus on extending FCR's routing mechanism to heterogeneous multi-agent collaboration.

VI. Limitations and Future Work

This work is subject to two primary limitations. First, the scope of FCR is currently confined to code generation tasks; its applicability to other reasoning domains, such as mathematical theorem proving, has not yet been explored. Second, the SCR's memory update mechanism relies on offline clustering, which prevents it from adapting to novel failure modes in real time. Future research will address these limitations by extending the FCR framework to handle multi-task settings and by designing a dynamic update mechanism for the SCR.

References

[1] M. Chen, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[3] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The eleventh international conference on learning representations*, 2022.

[6] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang *et al.*, "Aflow: Automating agentic workflow generation," *arXiv preprint arXiv:2410.10762*, 2024.

[7] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu *et al.*, "Autogen: Enabling next-gen llm applications via multi-agent conversations," in *First Conference on Language Modeling*, 2024.

[8] Y. Wang, Z. Xu, Y. Huang, X. Wang, Z. Song, L. Gao, C. Wang, X. Tang, Y. Zhao, A. Cohan *et al.*, "Dyflow: Dynamic workflow framework for agentic reasoning," *arXiv preprint arXiv:2509.26062*, 2025.

[9] J. Zhang, K. Cai, Q. Zeng, N. Liu, S. Fan, Z. Chen, and K. Wang, "Failure-driven workflow refinement," *arXiv preprint arXiv:2510.10035*, 2025.

[10] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 534–46 594, 2023.

[11] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634–8652, 2023.

[12] J. Su, Q. Lan, Y. Xia, L. Sun, W. Tian, T. Shi, X. Song, and L. He, "Difficulty-aware agentic orchestration for query-specific multi-agent workflows," *arXiv preprint arXiv:2509.11079*, 2025.

[13] G. Zhang, Y. Yue, X. Sun, G. Wan, M. Yu, J. Fang, K. Wang, T. Chen, and D. Cheng, "G-designer: Architecting multi-agent communication topologies via graph neural networks," *arXiv preprint arXiv:2410.11782*, 2024.

[14] C. Zheng, J. Chen, Y. Lyu, W. Z. T. Ng, H. Zhang, Y.-S. Ong, I. Tsang, and H. Yin, "Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming," *arXiv preprint arXiv:2505.22967*, 2025.

[15] S. Xu, J. Zhang, S. Di, Y. Luo, L. Yao, H. Liu, J. Zhu, F. Liu, and M.-L. Zhang, "Robustflow: Towards robust agentic workflow generation," *arXiv preprint arXiv:2509.21834*, 2025.

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

[17] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica, "Livecodebench: Holistic and contamination free evaluation of large language models for code," *arXiv preprint arXiv:2403.07974*, 2024.