Instructions for *ACL Proceedings

Anonymous ACL submission

Abstract

AI scientists have shown that they are capable of handling end-to-end research workflows. However, current research agents often rely heavily on large frontier models and monolithic execution loops that tightly couple ideation and implementation leading to poor failure attribution and unstable search behavior. Smaller models in particular struggle with long-horizon research tasks for the same reasons. We propose DeCoRe: Decoupled Collaborative Research Framework for Robust AI Scientific Workflows, that decouples ideation and codification into isolated iterative loops. The framework introduces components including an Idea Map, Task Analyzer, Semantic and Syntactic Smoke Tests for early-stage validation, and an Analysis and Feedback module. Together, these mechanisms improve performance across all metrics and reduce unnecessary experimentation. Experiments on four challenging repository-grounded tasks from the ResearchGym benchmark show that DeCoRe substantially improves the capability of GPT-5-based research agents, achieving competitive performance with stronger frontier proprietary agents while maintaining significantly lower variance and higher execution success rates across runs. These results suggest that structured iterative workflows can substantially amplify the effectiveness of weaker LLMs on complex autonomous research tasks.

1 Introduction

Large language models (LLMs) have emerged as powerful tools for augmenting the iterative research workflows that underpin modern machine learning research, from hypothesis generation and experiment design to implementation and evaluation. (Lu et al., 2024a; Yamada et al., 2025a; Gridach et al., 2025; Reddy and Shojaee, 2025). Recent systems such as AI Scientist-v2 (Yamada et al., 2025a), AI-Researcher (Tang et al., 2025), and DeepScientist (Weng et al., 2025b) demonstrate that LLM-based agents can contribute significantly across multiple stages of the research lifecycle. As AI augments end-to-end research — from ideation through execution — the resulting pipeline complexity makes it increasingly difficult to diagnose failure points or understand where a research direction derailed (Lu et al., 2024a; Yamada et al., 2025a). Hence, systems that enable rapid detection of gaps and failures in AI-assisted research pipelines to reduce cognitive burden have become essential. For more details and related work refer Appendix B.

Thus, current AI-assisted research workflows face following key challenges: (i) They often rely heavily on frontier-scale models and lack clear stage separation, making it difficult to determine whether failures stem from flawed ideas or implementation errors; (Trehan and Chopra, 2026; Seo et al., 2026); (ii) they rely mainly on either empirical execution metrics or heuristic LLM-based judgments for iterative refinements with limited mechanisms that combine semantic reasoning with execution-grounded validation. This provides limited insights to access failure cases (Si et al., 2026; Liu et al., 2025).These limitations become especially pronounced for weaker models, where cascading implementation errors and weak execution-guided feedback significantly reduce the effectiveness of iterative search. Consequently, researchers often spend considerable time rerunning experiments and debugging AI-generated outputs without clear guidance how to fix the failures.

To address these challenges, we propose DeCoRe, a structured modular workflow that organizes the AI-assisted research cycle into two explicit, inspectable stages: an *Ideation Loop* and a *Codification Loop*. By separating research direction generation from code implementation, DeCoRe enables failures to be attributed to the appropriate stage (Table 1). The Ideation loop that generates candidate research directions using

	Iso	Repo	Rank	Sem	Fea	Goal	Feed,	Ana.	Emp,	Evo.	Human.	Robu.	Nov.	Repro.	Mem.	Rec.	Bud.
Idea Agents																	
spark	-	-	✓	✓	∼	✓	∼	✗	✗	∼	∼	∼*	∼	-	∼	✗	✗
Virsci	-	-	✓	✓	∼	✓	∼	✗	✗	∼	∼	∼*	∼*	-	✓	∼	✗
SciMon	✗	✗	∼	✓	✗	✓	∼	✗	✗	✓	∼	∼	∼*	-	✓	∼	✗
Coding Agents																	
Basic Agent	✗	✓	✗	∼	✗	✗	∼	∼	∼*	✗	∼	∼	✗	∼	✗	∼	✗
SWE-Agent	✗	✓	✗	∼	∼	∼	∼	✓	✓	✗	∼	∼	✗	✓	∼	∼	✗
DARS	✗	✓	∼	∼	∼	-	✓	✓	∼	∼	✗	∼	✗	✓	∼	∼	✓
Codex	∼	✓	✗	∼	∼	∼	∼	∼	∼	✗	✓	∼	✗	✓	✗	∼	✗
Claude Code	∼	✓	✗	∼	∼	∼	✓	∼	∼	✗	✓	∼	✗	✓	✗	∼	✗
Evolutionary and Self-Improving Systems																	
ShinkaEvolve	∼	∼	✓	✗	✓	∼	✗	∼	✓	✓	✗	∼	∼*	✓	∼	✓	✓
Darwin Godel Machine	∼	✓	✗	✗	∼	✗	∼	∼	✓	✓	✗	∼	∼*	✓	∼	✓	✗
REVERE	✗	✓	✗	✗	∼	∼	✓	∼	∼	∼	✗	✓	∼	✓	✓	✓	∼
Research Platforms and Infrastructure																	
R&D-Agent	✓	∼	✓	✓	∼	∼	∼	∼	✓	✓	✗	∼	∼	✓	✓	✓	✓
ResearchTown	∼	✗	∼	∼	∼	∼	∼	✗	✗	✗	∼	∼	∼	∼	✓	✗	∼
PiFlow	✓	✗	✓	∼	✗	✓	∼	✗	✓	✓	✗	✓	∼	∼	✓	✓	∼
End-to-End Scientific Agents																	
AI Scientist	✓	✓	✓	∼*	∼	✓	∼	∼	✓	✓	∼	∼	∼*	✓	∼	✓	∼
Agent Laboratory	✓	✓	∼	∼	∼	✓	✓	∼	✓	∼	✓	∼	✗	✓	∼	✓	∼
AI-Researcher	✓	✓	✓	∼	∼	✓	✓	✓	✓	∼	∼	✓	∼*	✓	∼	✓	✗
CycleResearcher	✓	✗	✓	∼	∼	∼	✓	∼	✗	✓	∼	∼	∼*	∼	∼	∼	✗
AI Scientist-v2	✓	✓	✓	∼	∼	✓	✓	✓	✓	✓	∼	∼	∼*	✓	∼	✓	✓
DeepScientist	✓	✓	✓	✓	✓	✓	∼	✓	✓	✓	∼	∼	∼*	✓	✓	∼	✓
AutoSOTA	✓	✓	✓	✓	✓	✓	✓	∼*	✓	✓	∼	∼	∼*	✓	✓	✓	✓
Decore	✓	✓	✓	✓	✓	✓	✓	∼*	✓	✓	✓	∼	∼*	✓	✓	✓	✓

Table 1: Comparative evaluation dimensions for autonomous scientific research agents. Columns denote: isolated and holistic iterative research loops (Iso.), repository/codebase compatibility (Repo.), idea ranking and prioritization (Rank), semantic viability checking (Sem.), feasibility awareness under practical constraints (Fea.), research goal orientation (Goal), open-ended feedback integration (Feed.), analytical capability over experimental outputs and logs (Ana.), empirical validation through execution and experimentation (Emp.), evolutionary idea refinement across iterations (Evo.), human collaboration support (Human.), robustness and generalization across domains and tasks (Robu.), explicit novelty generation mechanisms (Nov.), reproducibility of artifacts and experiments (Repro.), memory and knowledge accumulation (Mem.), failure recovery and self-correction (Rec.), and budget-constrained optimization (Bud.). Here, ✓ indicates explicit and operational support, ∼ the capability is partially supported with limitations or inconsistency; ∼* the capability is conceptually present but lacks robust implementation or validation; ✗ indicates the dimension is not supported, and – indicates the dimension is not applicable.

a *Task Analyzer*, an AI-augmented module that discovers promising improvement dimensions for the provided research goal, These dimensions initialize the *Idea Map*, a memory structure that stores explored trajectories, outcomes, and feedback and are expert-validated. The Codification Loop then translates validated ideas into executable experiments using isolated implementation instances. Both loops incorporate *smoke tests* that perform early sanity checks, filtering incoherent ideas and faulty implementations to isolate issues within their respective stages for targeted refinement or rejection. Finally, a structured *Analysis and Feedback* module converts empirical outcomes into actionable signals, distinguishing idea-level from implementation-level failures and updating the Idea Map to guide future iterations.

We evaluate DeCoRe on four open-ended, repository-grounded research tasks from ResearchGym benchmark(Garikaparthi et al., 2026) benchmark, selected for their complexity and the breadth of valid solution approaches: Continual Learning, Cross-Modal Retrieval, Improving Replay Buffer and Time Series Explanation. These tasks require coordinated ideation, implementation, and execution, providing a suitable testbed for analyzing structured iteration and failure attribution in agentic research systems. In three out of four tasks, DeCoRe substantially improves the capability of GPT-5-based research agents, achieving competitive performance with stronger frontier proprietary agents while maintaining significantly lower variance and higher execution success rates across runs

Our main contributions are as follows:

- We propose DeCoRe, a structured AI-assisted research workflow system that decouples

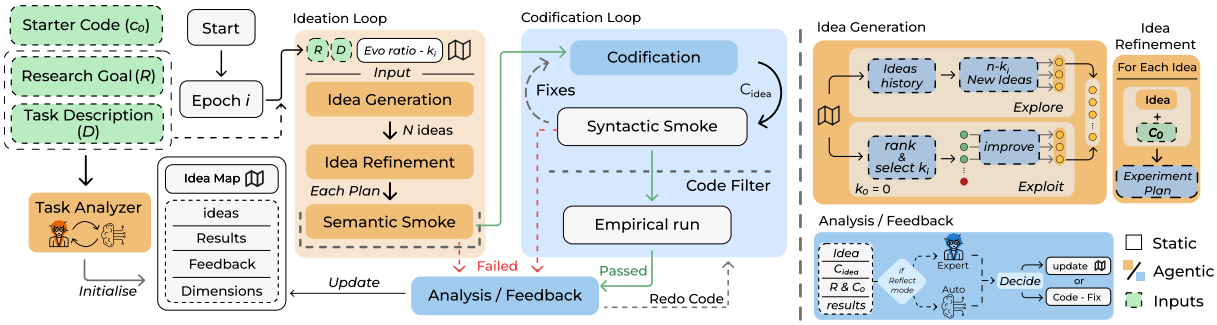

Figure 1: Illustration of **DeCoRe** that modularize autonomous research workflows into two decoupled iterative loops: (i) Ideation Loop performs exploration–exploitation based idea generation using a persistent Idea Map and Task Analyzer-guided dimensions, followed by an Idea Refinement module that expands candidate ideas into structured experiment plans and a Semantic Smoke Test for early feasibility filtering. (ii) Codification Loop converts validated plans into executable repository modifications, applies Syntactic Smoke Testing, and performs empirical execution. Structured Analysis/Feedback routes implementation failures back to codification and idea-level failures back to ideation, enabling execution-grounded iterative refinement across epochs.

ideation from implementation into two isolated, inspectable loops, enabling researchers to attribute failures to the correct stage and iterate more effectively without redundant full-pipeline reruns.

- We introduce ideation grounded in a persistent *Idea Map* that balances exploration of new research directions with exploitation of promising prior ones, helping researchers build systematically on accumulated experimental evidence across iterations.

- We demonstrate that structured iterative workflows can substantially amplify the capability of weaker models, specifically enabling GPT-5-based systems to achieve competitive performance with stronger frontier proprietary agents on complex repository-grounded research tasks while maintaining lower variance across runs.

2 DeCoRe

We present DeCoRe, a structured framework for autonomous iterative research loops designed to improve execution reliability and long-horizon refinement for weaker models. Existing agentic systems often entangle ideation, implementation, and execution within a single iterative workflow, making failure sources difficult to isolate and causing promising directions to collapse due to implementation instability. DeCoRe addresses this by separating conceptual refinement from implementation refinement through two decoupled processes: an Ideation Loop and a Codification Loop, connected through progressive validation and structured feedback routing.

DeCoRe begins with a human-guided *Task Analyzer* (see Appendix C.2 for details) that identifies promising improvement dimensions from the research goal $\mathcal{R}$, task description D (see Appendix D for example), capturing the kind of reasoning a researcher brings through knowledge of prior limitations, literature gaps, and domain-specific constraints. Importantly, the Task Analyzer works at the level of improvement dimensions rather than specific ideas, helping it avoid the specificity bias that leads many systems (Liu et al., 2025; Si et al., 2026; Gottweis et al., 2025) to prematurely fixate on narrow solutions instead of fully exploring the broader ideation space. Starting from these dimensions, DeCoRe runs epoch-based refinement where each epoch i generates n candidate ideas $\mathcal{I}$ along promising dimensions, checks them for semantic viability, implements them in isolation, evaluates them empirically, and feeds structured feedback into the next round of ideation. Figure 1 provides an overview of the full pipeline, and Algorithm 1 presents the complete workflow.

2.1 Ideation Loop

The Ideation Loop is responsible for generating and refining candidate research directions across epochs. Research agents often struggle with premature convergence during ideation, repeatedly generating narrow variations of previously explored directions while failing to maintain long-horizon search memory. To improve exploration quality and iterative refinement, DeCoRe structures ideation through accumulated search memory, evo-

lutionary exploration–exploitation, and progressive semantic validation.

Algorithm 1 DeCoRe – Full Pipeline

Require: $\chi = \{$ task description D , Research goal $\mathcal{R}$, starter code $C_0\}$, idea size n, budget B

Ensure: Best-performing idea ι^* and implementation c^*

$\mathcal{M} \leftarrow$ TASKANALYZER$(\mathcal{R},\ D)$ ▷ Initialise Idea Map
$\sigma \leftarrow (c,\ r,\ status,\ step)$ ▷ State with code, result, status $\in$ {pass, fail}, step $\in$ {semantic, code, result}
$k_0 \leftarrow 0$ ▷ Pure exploration in first epoch
for each epoch i **do**
  $\mathcal{I} \leftarrow$ IDEATION$(\mathcal{M}, \chi, n,\ k_i)$ ▷ Generate idea pool
  for each idea $\iota \in \mathcal{I}$ **do**
    $\sigma \leftarrow$ SEMANTICSMOKE(χ, ι) ▷ Checks viability
    if $\sigma.status =$ pass **then**
      $\sigma \leftarrow$ CODIFICATION$(\chi, \iota,\ \sigma,\ B)$
    end if
    $\mathcal{M}_i \leftarrow$ ANALYSISFEEDBACK$(\chi, \iota,\ \sigma)$
    $\mathcal{M} \cup \{\mathcal{M}_i\}$
  end for
  $k_{i+1} \leftarrow$ EVORATIO(i) ▷ increase exploitation
end for
return Best$(\iota^*,\ c^*)$ from $\mathcal{M}$

2.1.1 Task Analyzer and Idea Map

Research agents often struggle with premature convergence during ideation, repeatedly generating narrow variations of previously explored directions while failing to maintain long-horizon search memory. To improve exploration quality, DeCoRe initializes the ideation process using a *Task Analyzer* (see Appendix C.2 for details) that extracts high-level improvement dimensions from the $\mathcal{R}$ and task description D. Rather than directly generating concrete solutions, the Task Analyzer identifies broader axes of improvement such as architectural modifications, optimization strategies, or memory mechanisms etc., enabling more diverse exploration of the research space.

These dimensions then initialize the *Idea Map* $\mathcal{M}$(see Appendix C.3 for details), a persistent search memory that stores explored trajectories, empirical outcomes, failure annotations, and promising unexplored directions across epochs. By doing so it improves search continuity, reduces repeated exploration of previously unsuccessful directions, and supports more stable long-horizon iterative refinement.

2.1.2 Evolutionary Ideation

Building on the improvement dimensions and accumulated search memory maintained within the Idea Map, the Ideation Loop performs iterative refinement across epochs through an evolutionary exploration–exploitation process. At each epoch i, the loop generates a pool of n candidate ideas by balancing *exploration*, which generates $n - k_i$ new ideas along underexplored dimensions, with *exploitation*, which ranks and improves k_i directions retrieved from the *Idea Map*. This process is governed by an evolution ratio k_i that grows across epochs, shifting the balance from pure exploration toward increasing exploitation over time. Each idea candidate ι is further expanded into a structured experiment plan via an *Idea Refinement* module (Appendix C.9) which specifies repository modifications and experimental configuration required to evaluate the idea. (See Appendix Algorithm 2 and Algorithm 3 for the full exploration-exploitation schedule.)

2.1.3 Semantic Validation

Long-horizon ideation often produces infeasible, incoherent, or repository-incompatible experiment plans, leading to wasted implementation effort and unstable iterative search. To reduce invalid executions early, every plan undergoes a *Semantic Smoke Test* (Appendix C.4) which sanity checks for coherence, feasibility, and compatibility with the repository of the proposed idea in natural language space, with rejected ideas recorded in the Idea Map with failure annotations to guide future ideation. By filtering semantically invalid plans before implementation, DeCoRe reduces wasted experimentation and stabilizes long-horizon iterative refinement under constrained budgets.

2.2 Codification Loop

The Codification Loop is responsible for translating validated research directions into executable repository modifications and empirical experiments. While the Ideation Loop improves search quality and filters incoherent plans early, implementation instability remains a major bottleneck in long-horizon autonomous research workflows, where

Table 2: Performance comparison on Continual Learning (CL), Cross-Modal Retrieval (CMR), Improving Replay Buffer (IRB) and Time Series Explanation (TIM) tasks. We report average performance across runs, best-of-3 performance. Norm: Normalized score (see Appendix C.8 for details), *: results reported in ResearchGym (Garikaparthi et al., 2026). **Best Overall** , **Best within the group** performance.

	CL			CMR			IRB		TIM			Overall
Method	**Acc ↑**	**AAA ↑**	**Norm ↑**	**I2TR@1 ↑**	**T2IR@1 ↑**	**Norm ↑**	**Avg ↑**	**Norm ↑**	**CPD ↑**	**AUP/AUR ↑**	**Norm ↑**	**Norm ↑**
Baseline	86.30	90.72	0	83.50	69.90	0	3395	0	0.179	0.658	0	0
SOTA	88.01	92.54	1	86.80	70.30	1	4102	1	0.186	0.845	1	1
Average												
Frontier Proprietary Agents												
Codex*$_{(\text{GPT-5.4})}$	**85.00**	**90.00**	**−0.55**	84.20	66.70	−0.66	2995	−0.56	**0.114**	0.728	**−4.45**	**−1.55**
Claude Code*$_{(\text{OPUS-4-6})}$	84.70	89.0	−0.91	83.20	66.70	−0.94	**3348**	**−0.06**	0.103	**0.773**	−5.12	−1.76
Claude Code*$_{(\text{OPUS-4-5})}$	80.30	86.90	−2.74	**86.60**	**69.40**	**0.69**	2176	−1.72	0.102	0.719	−5.33	−2.27
GPT-5/5.2 Based Agents												
Codex*$_{(\text{GPT-5.2})}$	83.80	88.20	−1.40	84.30	69.40	0.07	178	−4.55	0.044	0.406	−10.31	−4.04
RG(GPT-5)*	33.80	45.60	−27.32	80.20	69.80	−0.90	580	−3.98	**0.131**	0.572	**−3.65**	−8.95
DeCoRe$_{(\text{GPT-5})}$	**83.90**	**90.10**	**−0.84**	**86.01**	**69.98**	**0.70**	**1583**	**−2.56**	0.116	**0.677**	−4.40	**−1.77**
Best@3												
Frontier Proprietary Agents												
Codex*$_{(\text{GPT-5.4})}$	**87.80**	**92.30**	**0.87**	**86.70**	69.40	0.73	**3630**	**0.33**	**0.122**	0.769	**−3.77**	**−0.46**
Claude Code*$_{(\text{OPUS-4-6})}$	87.70	90.90	0.45	86.60	**69.60**	**0.75**	3348	−0.06	0.104	**0.899**	−4.71	−0.89
Claude Code*$_{(\text{OPUS-4-5})}$	86.90	91.20	0.32	86.60	69.40	0.69	3014	−0.54	0.102	0.719	−5.33	−1.21
GPT-5/5.2 Based Agents												
Codex*$_{(\text{GPT-5.2})}$	86.00	90.07	−0.07	84.10	69.70	0.09	300	−4.38	0.049	0.415	−9.93	−3.57
RG*$_{(\text{GPT-5})}$	80.60	82.40	−3.92	80.20	70.00	−0.86	1407	−2.81	**0.172**	0.655	**−0.50**	−2.02
DeCoRe$_{(\text{GPT-5})}$	**87.30**	**92.15**	**0.69**	**86.92**	**70.36**	**1.05**	**1708**	**−2.38**	0.124	**0.711**	−3.78	**−1.10**

execution failures can cascade across iterations and obscure promising directions. To improve execution reliability, the Codification Loop utilizes isolated implementation instances, progressive syntactic validation, and structured feedback routing.

2.2.1 Isolated Codification

The Codification Loop translates validated experiment plans into executable repository modifications. For each candidate idea, DeCoRe initializes a fresh isolated codification instance (see Appendix C.5) against the original repository state C_0, preventing cross-idea contamination and reducing cascading implementation failures across iterations. Given the validated experiment specification, the codification agent generates repository-grounded code modifications, training configurations, and execution scripts required for empirical evaluation. (See Appendix A Algorithm 4 for codification loop).

2.2.2 Syntactic Validation

Code generation frequently produces malformed patches, runtime failures, dependency conflicts, or invalid execution pipelines that can destabilize iterative experimentation and waste computational budget. To prevent invalid executions, every generated implementation undergoes a *Syntactic Smoke Test* (See Appendix C.6) before empirical evaluation. The validation stage checks for implementation-level failures including compilation issues, runtime errors, missing dependencies, and repository compatibility constraints. Implementations that fail validation are routed through a bounded repair loop with targeted corrective feedback until either the implementation passes validation or a repair budget $\mathcal{B}$ is exhausted. Failed implementations that exceed the repair budget are forwarded directly to the Analysis and Feedback module. By filtering faulty implementations before execution, DeCoRe improves execution success rates and reduces wasted experimentation.

2.2.3 Analysis and Feedback Routing

Empirical execution metrics alone often provide weak signals for iterative refinement, making it difficult to determine whether poor outcomes arise from flawed research directions or unstable implementations. To improve failure attribution, DeCoRe incorporates a structured Analysis and Feedback module (See Appendix C.7 and Appendix A Algorithm 5 for details) that converts execution outcomes into actionable corrective signals. It operates over the execution state σ_i maintained for each component, containing the current implemented code, the step and status at which it succeeded or failed, and any empirical results if present. Given σ_i and the task context

$\chi = \{D, \mathcal{R}, C_0\}$, the module runs in one of two configurable modes: expert mode, where a human decision is informed by the provided analysis, or autonomous mode, where an LLM-as-judge reflects on the current context. In either case, outcomes are attributed to idea-level or implementation-level causes, producing a structured annotation M_i. If a failure is diagnosed as a straightforward implementation error, or a promising direction retains potential (e.g., hyperparameter adjustment or configuration changes), the component re-enters Codification for one additional loop with targeted feedback; the updated execution state then yields a refined M_i, which is recorded in $\mathcal{M}$ to guide subsequent ideation epochs.

Overall, DeCoRe combines search-space structuring, evolutionary ideation, progressive semantic and syntactic validation, isolated implementation refinement, and execution-grounded feedback routing within two decoupled iterative loops for ideation and codification. Together, these mechanisms reduce cascading failures, improve execution reliability, and enable weaker models to achieve more stable and competitive performance on complex repository-grounded research tasks under constrained compute and model budgets.

3 Experiment and Results

We evaluate DeCoRe on four open-ended repository-grounded research tasks from the ResearchGym(Garikaparthi et al., 2026) benchmark. These tasks require coordinated ideation, repository modification, empirical execution, and iterative refinement under constrained compute budgets, providing a challenging testbed for studying long-horizon autonomous research workflows. Our experiments aim to evaluate whether structured iterative workflows can improve the capability, stability, and execution reliability of weaker GPT-5-based research agents relative to monolithic agentic systems and stronger frontier proprietary agents.

3.1 Tasks and Evaluation

We evaluate DeCoRe on four repository-grounded research tasks from the ResearchGym benchmark (Garikaparthi et al., 2026): Continual Learning (CL), Cross-Modal Retrieval (CMR), Improving Replay Buffer (IRB), and Time Series Explanation (TIM) (see Appendix D for details). These tasks were selected for their long-horizon nature, repository-level implementation complexity, and broad space of valid solution strategies. Unlike constrained engineering benchmarks, the tasks require coordinated ideation, multi-file repository modification, empirical execution, and iterative refinement under limited computational budgets.

We adopt the task descriptions, starter repositories, and evaluation pipelines provided by ResearchGym, enabling standardized repository-grounded evaluation across all systems. Results are reported under an AI-assisted setting where improvement dimensions are initialized once using the Task Analyzer with expert guidance, while the remaining ideation, codification, validation, and feedback pipeline operates autonomously using LLM-based generation and judgment.

We compare against both frontier proprietary agents and GPT-5/5.2-based agentic systems. Frontier baselines include Claude Code (Opus-4.5/4.6) and Codex (GPT-5.4), while GPT-5/5.2-based baselines include RG-Agent, a unified ReAct-style research agent, and Codex (GPT-5.2). Beyond these baselines, we additionally attempted to integrate existing autonomous research systems specialized for scientific and engineering workflows. AI-Scientist-v2 (Yamada et al., 2025b) employs tree-based planning with parallel workers, but its prompting and execution assumptions were incompatible with repository-grounded tasks requiring modification of existing datasets, training pipelines, and evaluation protocols. Similarly, ML-Master (Liu et al., 2025) uses MCTS-based exploration optimized for MLE-Bench-style single-file engineering tasks, making it difficult to adapt to long-horizon repository-level experimentation involving coordinated multi-file modifications. In practice, both systems exhibited limited robustness under our task setting, supporting the need for workflows explicitly designed for repository-grounded autonomous iterative research.

CL evaluates rehearsal-free continual adaptation using final accuracy (Acc) and average incremental accuracy (AAA). CMR evaluates retrieval robustness under query shift using image-to-text retrieval at rank 1 (I2TR@1) and text-to-image retrieval at rank 1 (T2IR@1). IRB evaluates average (Avg) retrieval quality, while TIM evaluates reasoning-intensive experimentation through (CPD) and (AUP/AUR) metrics. In addition to task-specific metrics, we also report a normalized score (< 0: worse than baseline; $0 <$ and < 1: between baseline and the published SOTA, > 1: surpassing SOTA) (Appendix C.8 for details)

3.2 Evaluation Setup

We evaluate DeCoRe under a constrained-budget iterative search setting designed to study long-horizon autonomous refinement. For each task, the framework operates over 3 epochs with 5 candidate ideas generated per epoch, resulting in 15 explored research directions per run. This setup enables broader exploration and iterative refinement under fixed compute and time budgets compared to monolithic agentic systems that typically execute only a small (approx 3 to 4) of end-to-end trajectories. The Codification Loop is powered by GPT-5 and performs one implementation pass followed by up to 3 targeted repair attempts through the Syntactic Validation stage. Each candidate idea is implemented within an isolated repository instance initialized from the original repository state, preventing cross-idea contamination across iterations. Final task performance corresponds to the best successfully executed idea observed across all epochs, aligning with the evolutionary refinement nature of the framework where promising directions may emerge at different stages of search.

For each task, we report both average performance and Best@3 performance across three independent runs to account for stochasticity in ideation, implementation, and execution trajectories. All experiments are conducted under approximately fixed budget constraints of $10 and 12 hours per run.

3.3 Main Results

Table 2 summarizes the performance of DeCoRe across all four repository-grounded tasks. Across the evaluation, DeCoRe consistently improves the capability of GPT-5-based research agents, substantially outperforming GPT-5 baselines while remaining competitive with significantly stronger frontier proprietary agents. These gains are particularly pronounced in long-horizon settings requiring coordinated ideation, repository modification, empirical execution, and iterative refinement.

The strongest improvements are observed on Cross-Modal Retrieval (CMR), where DeCoRe achieves the best overall Best@3 performance, surpassing both frontier proprietary agents and the published SOTA in normalized score. Even in the average setting, DeCoRe remains competitive with frontier systems, suggesting that structured iterative refinement is highly effective for execution-intensive retrieval workflows where empirical feedback can reliably guide subsequent search iterations.

On **Continual Learning (CL)**, DeCoRe achieves performance competitive with frontier proprietary agents while substantially outperforming GPT-5 and GPT-5.2 based baselines. However, the average CL performance remains slightly below the strongest frontier models. We hypothesize that this gap arises from the greater optimization sensitivity of continual learning settings, where small implementation or configuration changes can amplify catastrophic forgetting and significantly affect long-horizon adaptation behavior.

IR and TIM further highlight the generalization properties of the proposed workflow. On IR, DeCoRe consistently improves over GPT-5 and GPT-5.2 based baselines across both average and Best@3 settings, demonstrating that the proposed workflow improvements generalize across diverse repository-grounded research settings. TIM represents a substantially more challenging reasoning-intensive setting, where all evaluated systems, including frontier proprietary agents, obtain strongly negative normalized scores. While RG-Agent achieves the strongest single-run TIM performance in terms of (CPD), its overall behavior remains highly unstable across tasks and runs. In contrast, DeCoRe maintains more stable and consistent performance across the broader evaluation suite, suggesting that structured iterative workflows primarily improve robustness and refinement stability rather than isolated peak performance.

Beyond absolute performance, DeCoRe exhibits substantially more stable search behavior than monolithic baselines. RG-Agent shows extremely high variance across runs (e.g., ± 37.39 Acc on CL), whereas DeCoRe consistently produces high-performing executable solutions with significantly lower variance (e.g., ± 3.42 Acc on CL). The smaller gap between average and Best@3 performance further suggests improved execution reliability and more stable long-horizon refinement across epochs. Notably, despite using GPT-5 as the underlying codification model, DeCoRe achieves overall normalized performance competitive with frontier proprietary agents across both average and Best@3 settings, suggesting that structured iterative workflows can partially compensate for differences in base model capability on repository-grounded research tasks. Together, these results indicate that structured iterative workflows can substantially amplify the capability of weaker models

on complex repository-grounded autonomous research tasks.

3.4 Ablation Analysis

We perform ablation studies on the Continual Learning (CL) task to analyze the contribution of the major structural components in DeCoRe. Table 3 reports the number of successfully executed ideas, execution success rate, and normalized performance under a fixed budget setting. Across all ablations, removing individual components consistently reduces both execution reliability and overall performance, highlighting the importance of structured iterative refinement.

Table 3: Ablation Analysis on the CL task for a single run.

Variant	Successful Ideas ↑	Success Rate ↑	Norm Score ↑
DeCoRe	**14 / 15**	**93.3%**	**0.69**
w/o Semantic Smoke	12 / 15	80.0%	-1.59
w/o Syntactic Smoke	6 / 15	40.0%	-16.62
w/o Task Analyzer	13 / 15	86.7%	-5.71

Removing the Semantic Smoke Test significantly reduces execution efficiency, as infeasible or repository-incompatible experiment plans proceed into implementation and consume computational budget through failed execution attempts. In contrast, removing the Syntactic Smoke Test produces the largest degradation in execution reliability, reducing the number of successfully executed ideas from 14/15 to 6/15. This suggests that progressive implementation-level validation is critical for stabilizing long-horizon repository-grounded experimentation.We additionally observe that removing the Task Analyzer substantially degrades normalized performance despite maintaining relatively high execution success rates. This indicates that explicit search-space structuring and improvement-dimension guidance play an important role in maintaining ideation quality and preventing premature convergence toward weak research directions. Another notable observation from Table 3 is the exploration efficiency achieved under constrained budgets. Within approximately $10 and 12 hours per run, DeCoRe explores 15 candidate ideas while maintaining a 93.3% successful execution rate. In comparison, monolithic agentic systems typically explore only a small number of end-to-end trajectories under similar budget settings. Overall, the ablations suggest that the observed gains arise not from model scale alone, but from the combination of search-space structuring, progressive validation, isolated implementation refinement, and execution-grounded feedback routing within the proposed workflow.

4 Conclusion

We presented DeCoRe, a structured framework for autonomous iterative research loops that decouples conceptual refinement from implementation refinement through two interacting processes: an Ideation Loop and a Codification Loop. By combining search-space structuring, evolutionary ideation, progressive semantic and syntactic validation, isolated implementation refinement, and execution-grounded feedback routing, DeCoRe improves execution reliability and stabilizes long-horizon iterative refinement for repository-grounded research tasks. Across four challenging tasks from the ResearchGym benchmark, DeCoRe consistently amplifies the capability of GPT-5-based research agents, achieving competitive overall performance with stronger frontier proprietary agents while maintaining lower variance and higher execution reliability across runs. These results suggest that workflow structure and progressive validation can partially compensate for differences in base model capability, highlighting the importance of explicitly structured iterative refinement for autonomous research systems operating in complex repository-grounded environments. LLM assistance was taken in writing.

Limitations

DeCoRe still relies on strong underlying language models for repository understanding, code generation, and execution-grounded reasoning, and therefore does not fully eliminate the capability gap between smaller and frontier-scale models. The current framework also does not incorporate explicit literature retrieval or external scientific knowledge grounding, limiting its ability to reason over broader research context beyond the provided repository and task description. Additionally, our evaluation includes a limited set of autonomous research baselines, as several existing systems were difficult to adapt to repository-grounded long-horizon workflows. Finally, the evaluation is currently restricted to machine learning research tasks within the ResearchGym benchmark.

References

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering.

Tianyu Fan, Fengji Zhang, Yuxiang Zheng, Bei Chen, Xinyao Niu, Chengen Huang, Junyang Lin, and Chao Huang. 2026. Deepinnovator: Triggering the innovative capabilities of llms. *arXiv preprint arXiv:2602.18920*.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. , 610(7930):47–53.

Aniketh Garikaparthi, Manasi Patwardhan, and Arman Cohan. 2026. Researchgym: Evaluating language model agents on real-world ai research. *Preprint*, arXiv:2602.15112.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, and 1 others. 2025. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*.

Mourad Gridach, Jay Nanavati, Khaldoun Zine El Abidine, Lenon Mendes, and Christina Mack. 2025. Agentic ai for scientific discovery: A survey of progress, challenges, and future directions.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. 2025. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *Preprint*, arXiv:2506.16499.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024a. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024b. The ai scientist: Towards fully automated open-ended scientific discovery. *Preprint*, arXiv:2408.06292.

Jiacheng Miao, Joe R. Davis, Jonathan K. Pritchard, and James Zou. 2025. Paper2agent: Reimagining research papers as interactive and reliable ai agents. *Preprint*, arXiv:2509.06917.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *Preprint*, arXiv:2506.13131.

Chandan K Reddy and Parshin Shojaee. 2025. Towards scientific discovery with generative ai: Progress, opportunities, and challenges. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 28601–28609.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *Preprint*, arXiv:2501.04227.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2026. Paper2code: Automating code generation from scientific papers in machine learning. *Preprint*, arXiv:2504.17192.

Chenglei Si, Zitong Yang, Yejin Choi, Emmanuel Candès, Diyi Yang, and Tatsunori Hashimoto. 2026. Towards execution-grounded automated ai research. *Preprint*, arXiv:2601.14525.

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. 2025. Paperbench: Evaluating ai's ability to replicate ai research. *Preprint*, arXiv:2504.01848.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. 2025. Ai-researcher: Autonomous scientific innovation. *arXiv preprint arXiv:2505.18705*.

Dhruv Trehan and Paras Chopra. 2026. Why llms aren't scientists yet: Lessons from four autonomous research attempts. *Preprint*, arXiv:2601.03315.

Yixuan Weng, Minjun Zhu, Guangsheng Bao, Hongbo Zhang, Jindong Wang, Yue Zhang, and Linyi Yang. 2025a. Cycleresearcher: Improving automated research via automated review. *Preprint*, arXiv:2411.00816.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. 2025b. Deepscientist: Advancing frontier-pushing scientific findings progressively. *arXiv preprint arXiv:2509.26603*.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. 2025a. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. 2025b. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *Preprint*, arXiv:2504.08066.

Minjun Zhu, Yixuan Weng, Linyi Yang, and Yue Zhang. 2025. Deepreview: Improving llm-based paper review with human-like deep thinking process. *Preprint*, arXiv:2503.08569.

A Algorithms

We present the detailed algorithms underlying each module of DeCoRe. Together, these algorithms implement the two iterative loops described in Section 2: the Ideation Loop and the Codification Loop, connected through the Analysis and Feedback module. Each algorithm highlights the flow of information between modules, the role of the Idea Map in maintaining persistent research memory, and the mechanisms used to balance exploration with iterative improvement. Collectively, these procedures define the operational workflow of DeCoRe and illustrate how the system performs structured autonomous research under constrained computational and execution budgets.

A.1 Ideation: Evolutionary Idea Generation

At each epoch, the Ideation module generates a pool of n candidate ideas by balancing exploration of new directions with exploitation of previously promising ones. The exploitation count k_i grows across epochs, governed by the EvoRatio schedule (Algorithm 3), shifting the search from broad exploration in early epochs toward targeted refinement in later ones.

Algorithm 2 Ideation : Evolutionary Idea Generation

Require: Idea Map $\mathcal{M}$, pool size n, exploitation count k_i, Task context χ
Ensure: Idea pool $\mathcal{I}_i$

$\mathcal{I}_{\text{explore}} \leftarrow \textsc{GenerateNew}(n - k_i, \mathcal{M}, \chi)$
$\mathcal{I}_{\text{ranked}} \leftarrow \textsc{Rank}(\textsc{History}(\mathcal{M}))$
$\mathcal{I}_{\text{exploit}} \leftarrow \textsc{Improve}(\textsc{TopK}(\mathcal{I}_{\text{ranked}}, k_i), \chi)$
$\mathcal{I}_i \leftarrow \mathcal{I}_{\text{explore}} \cup \mathcal{I}_{\text{exploit}}$
return $\mathcal{I}_i$

A.2 EvoRatio: Exploration–Exploitation Schedule

The EvoRatio function determines how many of the n ideas in each epoch are generated via exploitation of past results versus fresh exploration. In the first epoch ($i = 0$), the system explores purely with no exploitation. In subsequent epochs, the exploitation count grows as a function of epoch number, capped at a maximum fraction c of the idea pool. The hyperparameters a, b, and c controlling this schedule are detailed in Appendix C.1.

Algorithm 3 EvoRatio k (refer Appendix C.1 for hyperparamters a,b and c).

Require: Epoch number i, ideas per epoch n.
Ensure: Number of exploitative ideas k_i

if $i = 0$ **then**
  $k \leftarrow 0$
else
  $k \leftarrow \min(a + b(i/2),\ c)$
end if
$k_i \leftarrow \lfloor n \cdot k \rfloor$
return k_i

A.3 Codification with Empirical Verification

Once an idea passes the Semantic Smoke Test, it enters the Codification Loop. A fresh agent instance is initialized from the starter code C_0 and iteratively refines the implementation until it passes the Syntactic Smoke Test or exhausts the repair budget B. Implementations that pass syntactic validation proceed to empirical execution; those that exhaust the budget are forwarded to Analysis and Feedback with a failure status, so that the failure can be attributed and recorded in the Idea Map.

Algorithm 4 Codification with Empirical Verification

Require: Task Context χ, Idea ι, execution state σ, budget B
Ensure: Updated σ with $(c,\ r,\ \mathit{status},\ \mathit{step}{=}\texttt{code})$

$c \leftarrow \chi.C_0$ ▷ Initialize starter code
$e \in \varnothing$ ▷ Initially No errors
while $B > 0$ **do**
  $c \leftarrow \textsc{Codify}(\chi, \iota,\ c,\ e)$ ▷ Plan + latest code + syntactic error
  $c,\ ok,\ e \leftarrow \textsc{SyntacticSmoke}(c)$ ▷ Returns fixed code, pass/fail, and error e
  if ok **then**
    $r \leftarrow \textsc{EmpiricalRun}(c)$
    return $\sigma.\{c,\ r,\ \mathit{status}{=}\texttt{pass},\ \mathit{step}{=}\texttt{code}\}$
  end if
  $B \leftarrow B - 1$
end while
return $\sigma.\{c,\ \varnothing,\ \mathit{status}{=}\texttt{fail},\ \mathit{step}{=}\texttt{code}\}$ ▷ Return latest code even on failure

A.4 Analysis and Feedback

After execution, the Analysis and Feedback module interprets the execution state σ and decides the appropriate next action. Depending on the configuration, a human expert or an LLM judge analyzes

the outcome and determines whether the failure is attributable to the idea or to the implementation. If the failure is deemed fixable at the code level, the idea is augmented with targeted feedback and re-enters Codification for one additional repair pass. Regardless of the action taken, a structured memory entry M_i is produced by a Curator module and recorded in the Idea Map $\mathcal{M}$ to inform future ideation epochs.

Algorithm 5 Analysis and Feedback

Require: Task Context χ, Idea ι, execution state σ, budget B

Ensure: Updated memory $\mathcal{M}$ and resolved model output M_i, configuration Expert or LLM $\mathcal{L}$

$(f, a) \leftarrow \mathcal{L}_{judge}(\chi, \iota, \sigma)$ ▷ Human expert or LLM judge analyzes execution and decides action

if $a =$ `fix-code` **then**

  $\iota \leftarrow \iota + f$

  $\sigma_{\text{redo}} \leftarrow \textsc{Codification}(\chi, \iota, \sigma, B)$ ▷ Regenerate code using feedback

  $\sigma \leftarrow \sigma \cup \sigma_{\text{redo}}$ ▷ Merge previous and regenerated execution states

end if

$M_i \leftarrow \textsc{Curator}(\chi_\iota, \iota, \sigma)$ LLM curator generates structured information to be added to the Idea Map M_i

B Related Works

Early AI-assisted research systems focused on supporting individual stages of the scientific workflow. Systems such as CycleResearcher (Weng et al., 2025a), DeepReview (Zhu et al., 2025) and DeepInnovator(Fan et al., 2026) assist with tasks including writing, reviewing, hypothesis generation, and literature synthesis.Engineering-Centric Autonomous Systems. Benchmarks and replication-oriented frameworks such as PaperBench(Starace et al., 2025) and Paper2Agent(Miao et al., 2025), emphasize reproducing published methodologies, while systems like Agent Laboratory(Schmidgall et al., 2025) and MLE-Bench(Chan et al., 2024) focus primarily on script level machine learning engineering tasks, rather than long-horizon interaction with complex research repositories. Related approaches such as AlphaTensor(Fawzi et al., 2022) and AlphaEvolve(Novikov et al., 2025) perform large-scale optimization over predefined search spaces, but are not designed for open-ended autonomous scientific discovery involving iterative hypothesis generation and conceptual refinement.

More recent frameworks, such as AI Scientist(Lu et al., 2024b)(Yamada et al., 2025b) demonstrate that LLM agents can independently conduct substantial portions of the research process, including generating hypotheses, implementing experiments, and producing research artifacts. DeepScientist(Weng et al., 2025b) and (Gottweis et al., 2025), further introduce iterative evaluation and goal-directed exploration strategies. However, they lack clear stage separation.

C Module Details and Setup

C.1 Setup

All systems operate under the ResearchGym evaluation protocol: starter repository, task description, and grading script as inputs, constrained to $10 in LLM API costs and 12 hours on a single NVIDIA A100 (80GB VRAM). For RG-Agent we report best@3 over three seeds as provided by benchmark authors. For EvoRatio k we set $a = 0.5$, $b = 0.1$ and $c = 0.8$.

C.2 Task Analyzer

The Task Analyzer is a lightweight reasoning module that processes the research goal $\mathcal{R}$, task description D, and evaluation protocol to identify the key improvement dimensions relevant to the task. The module extracts a structured representation of desirable research properties, which is then inspected and refined by a human in the loop. These extracted dimensions initialize the Idea Map $\mathcal{M}$ and guide subsequent ideation by conditioning both exploration and exploitation toward research directions aligned with the task objectives. By grounding idea generation in task-specific dimensions rather than unconstrained brainstorming, the Task Analyzer reduces irrelevant exploration and improves the consistency of iterative refinement across epochs. We illustrate a few representative dimensions extracted by the Task Analyzer (e.g., rehearsal-free learning, inference efficiency, end-to-end optimization for the Continual Learning task); the complete set of dimensions used in our experiments are shown below :

C.3 Idea Map

The Idea Map $\mathcal{M}$ is a persistent data structure that accumulates the exploration history of DeCoRe across epochs. It is initialized by the Task Analyzer, which identifies the key improvement dimensions

Task Analyzer output for CL Task

n class-incremental learning with foundation models, an ideal solution should simultaneously satisfy three core dimensions:

1. **Rehearsal-free learning**: avoid storing past data to ensure scalability
2. **Inference efficiency**: avoid task-specific selection or growing components at test time
3. **End-to-end optimization**: jointly optimize all parameters for continual learning objectives

relevant to the research goal (e.g., task-agnostic inference, end-to-end optimization). As epochs progress, the Idea Map is enriched with:

- explored idea directions and their associated experiment plans;
- empirical results and structured feedback from the Analysis module;
- failure annotations from both the Semantic Smoke Test and post-analysis;
- the ranking and improvement history from the exploitation branch.

The Idea Map serves three roles: (i) it prevents redundant exploration by making the system aware of previously tried directions; (ii) it grounds exploitation by providing a ranked history of promising ideas for refinement; and (iii) it encodes failure patterns that the exploration branch actively avoids when generating new ideas.

C.4 Semantic Smoke Test(Idea Filter)

Before implementation begins, each experiment plan undergoes a Semantic Smoke Test, which evaluates the proposed idea directly in natural language space. This stage validates whether the proposed modification is logically coherent, sufficiently distinct from previously explored directions in the Idea Map, and feasible given the constraints of the task and starter codebase C_0. Ideas that fail this test are rejected and annotated in the Idea Map guiding the search in future epochs and also preventing the system from spending computation on codifying ideas that are flawed at a conceptual level.

C.5 Codification Agent

The codification agent in DeCoRe is a lightweight ReAct-style agent equipped with a controlled

Task Analyzer output for CMR Task

n cross-modal retrieval, an ideal solution should simultaneously satisfy four core dimensions:

1. **Cross-Modal Retrieval Performance**: Measures how effectively the method improves retrieval accuracy across modalities under real-world conditions.
2. **Robustness to Query Distribution Shift**: Captures the ability to maintain performance when incoming queries differ from the source distribution.
3. **Intra-Modality Uniformity**: Reflects how well the method preserves discrimination among queries within the same modality in the shared space.
4. **Inter-Modality Alignment (Modality Gap Control)**: Evaluates how effectively the method maintains alignment between query and gallery modalities despite distribution shifts.

Semantic Smoke Output

Idea id: 0

Feasiblilty: True

Reason: Coherent and implementable by adding a consistency loss and using existing LN-only adaptation hooks; requires moderate code changes in forward paths and TTA wrappers without violating constraints.

toolset for file inspection, patch application, and code modification. A fresh instance is spawned for every idea, ensuring complete isolation between ideas and preventing cross-idea contamination. Each instance operates under a strict per-idea budget that limits real-time usage, number of tool calls, and token consumption, enforcing predictable and interpretable behavior. The agent functions in two modes: a coding mode, where it translates the refined idea into concrete code edits during the initial implementation pass, and a fixing mode, where it performs targeted, minimal corrections in response to error signals returned by the execution environment. These modes combined allow the agent to handle both structured code gen-

eration and localized repair without overwriting unrelated logic. By design, the codification agent remains stateless, bounded, and fully specialized to a single idea, enabling precise failure attribution and consistent implementation quality across research epochs.

C.6 Syntactic Smoke Test (Code Filter).

After codification, the generated implementation undergoes a Syntactic Smoke Test before empirical execution. This stage validates repository-level correctness by detecting implementation failures such as syntax errors, invalid imports, unresolved dependencies, interface mismatches, configuration inconsistencies, and runtime initialization failures. When errors are detected, the corresponding codification instance re-enters a targeted repair phase within the Codification Loop, where localized fixes are applied without modifying unrelated repository components. By filtering broken implementations prior to expensive execution, the Syntactic Smoke Test improves execution stability and reduces wasted computational runs..

C.7 Analysis and Feedback

After execution, the Analysis and Feedback module performs structured post-analysis of the execution state σ for each idea ι. Unlike prior systems that rely solely on raw metric scores (e.g., a single accuracy number), this module generates rich, attributable feedback signals that distinguish between idea-level and implementation-level failures — a core design principle of DeCoRe.

The module operates in one of two configurable modes. In **expert mode**, a human reviewer is presented with the structured analysis of the execution state and makes the final routing decision. In **autonomous mode**, an LLM-as-judge reflects on the task context χ, the idea ι, and the execution state σ to determine the appropriate action. In both modes, the outcome is one of two actions: if the failure is attributed to a straightforward implementation error or a promising direction that requires minor adjustment (e.g., hyperparameter tuning or configuration changes), the idea re-enters the Codification Loop with targeted feedback for one additional repair pass. If the failure reflects a fundamental idea-level shortcoming, it is recorded in the Idea Map $\mathcal{M}$ with a failure annotation to inform future ideation epochs. Regardless of the action taken, a structured memory entry M_i is produced by the Curator and added to $\mathcal{M}$ to guide subsequent epochs.

C.8 Normalized Score

For tasks with a single evaluation metric, normalized performance is computed as:

$$S_{\text{norm}} = \frac{S_{\text{agent}} - S_{\text{baseline}}}{S_{\text{SOTA}} - S_{\text{baseline}}}$$

where S_{agent} denotes the score achieved by the evaluated agent and S_{SOTA} corresponds to the score reported by the original paper.

For tasks containing multiple primary subtasks or evaluation metrics, we compute normalized performance using the ratio between the average agent score and the average published reference score across subtasks:

$$S_{\text{norm}} = \frac{\frac{1}{m}\sum_{i=1}^{m}(S_{\text{agent}}^{(i)} - S_{\text{baseline}}^{(i)})}{\frac{1}{m}\sum_{i=1}^{m}(S_{\text{SOTA}}^{(i)} - S_{\text{baseline}}^{(i)})}$$

where (m) denotes the number of evaluated subtasks. A normalized score below 0 indicates performance worse than the benchmark baseline, scores between 0 and 1 indicate performance between the baseline and the published SOTA solution, and scores greater than 1 indicate performance surpassing the reported SOTA result.

C.9 Idea Refinement Module

For each idea ι that emerges from the Ideation Loop, Idea Refinement Module generates a structured experiment plan by conditioning on both the idea and the starter code C_0. The experiment plan specifies the concrete modifications to be made to C_0, the evaluation protocol, and the expected outcome. This plan serves as the contract between the Ideation Loop and the Codification Loop, ensuring that abstract research ideas are grounded in an actionable implementation specification before any code generation begins.

Idea Before Refinement

Name: Dual-Branch Test-Time Entropy + Alignment (DB-TEA)

Core claim: Combine entropy-minimized feature sharpening with contrastive alignment to the gallery prototypes during test-time adaptation to improve robustness under query shift without collapsing cross-modal alignment.

Mechanism: Maintain two branches at test time: (i) a prediction-sharpening branch minimizes per-batch conditional entropy on retrieval logits for the query vs. gallery embeddings; (ii) an alignment branch pulls the query embedding toward a running EMA prototype of its top-k nearest gallery embeddings using a temperatured InfoNCE loss with gallery as positives and hard-negatives from other clusters. Optimize a weighted sum $L = L_{entropy} + L_{align}$ with small learning rates on only layernorm/scale parameters in the encoders and the projection heads.

Risk: Prototype bias may overfit to incorrect nearest neighbors early, harming rare concepts.

Dimension Alignment: Targets Cross-Modal Retrieval Performance, Robustness to Query Distribution Shift, Inter-Modality Alignment; lightly impacts Intra-Modality Uniformity via hard-negative mining.

Idea After Refinement

Name: Dual-Branch Test-Time Entropy + Alignment (DB-TEA)

Core claim: Combine entropy-minimized feature sharpening with contrastive alignment to the gallery prototypes during test-time adaptation to improve robustness under query shift without collapsing cross-modal alignment.

Mechanism: Maintain two branches at test time: (i) a prediction-sharpening branch minimizes per-batch conditional entropy on retrieval logits for the query vs. gallery embeddings; (ii) an alignment branch pulls the query embedding toward a running EMA prototype of its top-k nearest gallery embeddings using a temperatured InfoNCE loss with gallery as positives and hard-negatives from other clusters. Optimize a weighted sum $L = L_{entropy} + L_{align}$ with small learning rates on only layernorm/scale parameters in the encoders and the projection heads.

Risk: Prototype bias may overfit to incorrect nearest neighbors early, harming rare concepts.

Dimension Alignment: Targets Cross-Modal Retrieval Performance, Robustness to Query Distribution Shift, Inter-Modality Alignment; lightly impacts Intra-Modality Uniformity via hard-negative mining.

Detailed:[Experiment]:We add a lightweight per-query gating module that reweights encoder features before the projection head to down-weight corruption-sensitive subspaces. At test time only the gating parameters are adapted online via entropy-minimization (Tent) without touching backbone weights, keeping gallery features fixed and preserving alignment. We evaluate on both CLIP and BLIP backbones; the primary metric targeted is Recall@1 under QS/QGS by reducing retrieval entropy while maintaining intra-modality uniformity. Hyperparameters: gate hidden width 128, L1 sparsity 1e-4, EMA smoothing 0.9, learning rate 3e-4 (same as existing)

D Tasks

D.1 Continual Learning

This task is based on SD-LoRA (ICLR 2025 Oral), which decomposes low-rank updates into magnitude (scalar coefficients) and direction (normalized matrices), freezing past directions while training only magnitudes and the current task's new direction.

Task Description

Research Goal Continual Learning (CL) with foundation models has emerged as a promising paradigm, but existing prompt-based and Low-Rank Adaptation-based (LoRA-based) methods have limitations. These methods often require expanding a prompt or LoRA pool, or retaining samples of previous tasks for rehearsal. This poses significant scalability challenges as the number of sequential tasks grows. Current CL methods with foundation models often fail to satisfy three desirable properties simultaneously: being rehearsal-free, maintaining inference efficiency, and allowing for end-to-end optimization of all parameters. The reliance on growing prompt/LoRA pools compromises inference scalability, while storing samples from previous tasks is not scalable in resource-constrained or large-scale settings. A new method is needed to address these limitations and achieve a more scalable and practical solution for continual learning.

Experimental Settings

- Evaluation Benchmarks: ImageNet-R, ImageNet-A, CIFAR-100, CUB-200.
- Task Splits:
  - ImageNet-R: 5 tasks (40 classes/task), 10 tasks (20 classes/task), or 20 tasks (10 classes/task).
  - ImageNet-A: 10 tasks (20 classes/task).
  - CIFAR-100: 10 tasks (10 classes/task). [Primary Sub-task]
  - CUB-200: 10 tasks (20 species/task).
- Foundation Models: ViT-B/16 (pre-trained on ImageNet-21K and fine-tuned on ImageNet 1K).
- Metrics: Average accuracy (Acc), Average anytime accuracy (AAA).
- Evaluation Setup: Results should be reported as mean across three runs with standard errors (use seeds 1992- 1994).

Datasets

- ImageNet-R
- ImageNet-A
- CIFAR-100
- CUB-200

Evaluation Metrics

- *Acc*: The Acc metric measures the overall performance by computing the average accuracy across all N tasks upon the completion of CL.
- *AAA*: AAA further accumulates the average accuracy of all encountered tasks after training on each new task.

D.2 Cross Modal Retrieval

This task is based on a method addressing query shift in cross-modal retrieval, which forms query candidate pairs, selects source-domain-like pairs using intra-modality uniformity and inter-modality gap scores, then applies a joint objective combining uniformity loss, gap rectification, and noise-robust adaptation, updating only normalization layers online. The approach is designed to improve retrieval robustness under distribution shifts where query samples differ significantly from the source-domain training distribution.

Task Description

Research Goal The success of most existing cross-modal retrieval methods heavily relies on the assumption that given queries follow the same distribution as the source domain. However, this assumption is easily violated in real-world scenarios due to the complexity and diversity of queries, leading to the query shift problem. Query shift refers to an online query stream originating from a domain that follows a different distribution than the source, causing significant performance degradation. In real-world applications like search engines, users may have diverse cultural backgrounds or personal preferences, resulting in online queries from scarce or highly personalized domains. These out-of-domain queries violate the identical distribution assumption that pre-trained models rely on. Consequently, existing cross-modal retrieval models fail to handle this query shift and suf fer significant performance drops, necessitating an online adaptation method to address this problem.

Experimental Settings

Source Models: CLIP (ViT-B/16) and BLIP (ViT-B/16, ViT-L/16).

Datasets & Settings:

- **Query Shift (QS)**: Queries have a different distribution from the gallery. Benchmarks created are COCO-C and Flickr-C, built from COCO and Flickr by adding 16 image corruption types (Noise, Blur, Weather, Digital) and 15 text corruption types (character, word, sentence-level) at various severity levels.
- **Query-Gallery Shift (QGS)**: Both query and gallery samples come from distributions different from the source. Datasets include Fashion-Gen (e-commerce), CUHK-PEDES and ICFG-PEDES (person Re-ID) **[Primary Sub-task]**, and COCO, Flickr **[Primary Sub-task]**, Nocaps (natural image).

Evaluation Metrics

- *T2IR@1*(Text-to-image retrieval)
- *I2TR@1* (Image-to-text retrieval)

Hints

- You are expected to setup the datasets and weights directories and populate using the provided links in the README.md file.
- You can use the gdown library to directly download google drive contents, and git clone for studying relevant repositories.

E Prompts

Idea Generation For Epoch 0

System Prompt

You are a research scientist. You will be given a research task with baselines and their scores.Your job is to propose $\{n\}$ distinct ideas that could beat the baselines and current SOTA.

$\{task_md\}$

$\{repo_str()\}$

[Improvement Dimensions] The following dimensions have been identified as promising directions:$\{rubrics\}$

These dimensions are core suggestions for achieving the research goal, not requirements. If you identify a more fundamental weakness in the baselines, or see an opportunity to combine insights from adjacent fields, pursue that in parallel. The only hard constraint is that your ideas must be testable within the existing evaluation framework.
For each of the n ideas, provide:

1. Name : A short descriptive label.
2. Core Claim : One sentence stating what this idea changes and why it should improve performance.
3. Mechanism : 2-4 sentences explaining the technical approach. Be specific about what computation changes, (for eg. what loss term is added, or what architectural modification is made). Do not describe about code changes or file-level edits.
4. Reasoning : Explain your rationale for proposing the idea and why it outperforms the baselines, including which baseline limitations it addresses and referencing specific scores or behaviors if relevant.
5. Risk : The single biggest reason this might fail.
6. Dimension Alignment : Which rubric dimensions does this target? Write "NONE - addresses [description]" if it targets something outside the provided dimensions.

Guidelines

- Diversity matters more than polish. Propose ideas that differ in mechanism. (for eg. not just in hyperparameter choices.)
- At least one idea should challenge an assumption shared by all baselines rather than incrementally extending them.
- You may draw inspiration from techniques in other domains (e.g., applying ideas from NLP to RL, or from physics-informed methods to time series). When you do, state the source domain and why the transfer is justified.
- Do not propose implementation details, file edits, or hyperparameter values. That is a later stage.
- Do not hedge every idea. State your claims directly. The risk field exists for caveats.

Constraints

- Do NOT suggest ideas that might need to modify grading related for eg. ‘grading/*‘ or task related for eg. ‘task_description.md‘.
- Do not suggest changes about the metric calculation logic, you can change the training logic

Evolutionary Exploit Idea Generation

System Prompt

You are a research scientist. You will be given a research task with baselines and their scores. Your job is to propose $\{n\}$ improvement to working ideas that could beat the baselines and current SOTA.

$\{task_md\}$

Below are prior successful ideas and their outcomes (primary metric shown):

$\{prev_idea_prompt\}$

They worked and ranked highest among the all the ideas implemented, may or maynot exceed the baseline for this task. Propose $\{n\}$ improved ideas that improve on these successes (e.g., combine compatible components, adjust objectives/regularization/curricula) while keeping changes minimal.

[Improvement Dimensions] The following dimensions have been identified as promising directions:$\{rubrics\}$

These dimensions are core suggestions for achieving the research goal, not requirements. If you identify a more fundamental weakness in the baselines, or see an opportunity to combine insights from adjacent fields, pursue that in parallel. The only hard constraint is that your ideas must be testable within the existing evaluation framework.
For each of the n ideas, provide:

1. Name : A short descriptive label.
2. Core Claim : One sentence stating what this idea changes and why it should improve performance.
3. Mechanism : 2-4 sentences explaining the technical approach. Be specific about what computation changes, (for eg. what loss term is added, or what architectural modification is made). Do not describe about code changes or file-level edits.
4. Reasoning : Explain your rationale for proposing the idea and why it outperforms the baselines, including which baseline limitations it addresses and referencing specific scores or behaviors if relevant.
5. Risk : The single biggest reason this might fail.
6. Dimension Alignment : Which rubric dimensions does this target? Write "NONE - addresses [description]" if it targets something outside the provided dimensions.

Constraints

- Do NOT suggest ideas that might need to modify grading related for eg. ‘grading/*‘ or task related for eg. ‘task_description.md‘.
- Do not suggest changes about the metric calculation logic, you can change the training logic
- Avoid any similar ideas to the following experiments that have been proposed before: $\{cache_str\}$

Evolutionary Explore Idea Generation

System Prompt

Below are representative prior ideas and their results: $\{prev_ideas_prompt\}$.

You are a research scientist. Propose $\{num_ideas\}$ new ideas that are distinct from prior attempts, prioritizing novelty and simplicity. You have to generate the experiment ideas so that they are likely to beat the baselines and current SOTA.

$\{task_md\}$

[Improvement Dimensions] The following dimensions have been identified as promising directions:$\{rubrics\}$

These dimensions are core suggestions for achieving the research goal, not requirements. If you identify a more fundamental weakness in the baselines, or see an opportunity to combine insights from adjacent fields, pursue that in parallel. The only hard constraint is that your ideas must be testable within the existing evaluation framework.
For each of the $\{num_ideas\}$ ideas, provide:

1. Name : A short descriptive label.
2. Core Claim : One sentence stating what this idea changes and why it should improve performance.
3. Mechanism : 2-4 sentences explaining the technical approach. Be specific about what computation changes, (for eg. what loss term is added, or what architectural modification is made). Do not describe about code changes or file-level edits.
4. Reasoning : Explain your rationale for proposing the idea and why it outperforms the baselines, including which baseline limitations it addresses and referencing specific scores or behaviors if relevant.
5. Risk : The single biggest reason this might fail.
6. Dimension Alignment : Which rubric dimensions does this target? Write "NONE - addresses [description]" if it targets something outside the provided dimensions.

Constraints

- Do NOT suggest ideas that might need to modify grading related for eg. ‘grading/*‘ or task related for eg. ‘task_description.md‘.
- Do not suggest changes about the metric calculation logic, you can change the training logic
- Avoid any similar ideas to the following experiments that have been proposed before: $\{cache_str\}$

Coding Agent

System Prompt

You are a helpful agent attempting to codify an idea provided to you on a research repository.

To achieve this task you have following instructions:

- You have several functions available to help with solving the task. Each message may perform one function call. You will see the result of the function right after sending the message. If you need to perform multiple actions, you can always send more messages with subsequent function calls.
- Make sure you finish the task in alloted time, so plan accordingly.
- You MUST make changes to run.ps1 file inorder to implement your changes as run.ps1 is executed by a seperate executor. Hence, you must make changes to run.ps1 file like chaniging the config file to invoke the changes.
- Do some reasoning before your actions, describing what function calls you are going to use and how they fit into your plan.
- Use as much of the available time as possible to refine and optimize your solution: Your goal is to impliment the idea, not just to submit your first working solution. Keep working until the time limit expires.
- The file read tool is paginated, so you will need to call it multiple times to make sure that you have read all the relevant parts of a paper.
- This is a multi-step task. DO THINGS BIT BY BIT by slowly building up your solution with multiple tool calls, don't try to do everything in one go. Just reason about what is the best next step and execute it. Then take the next step after you receive the output of the previous step.

Guidelines

- You should try and create a clean codebase. When it makes sense you should edit/overwrite files rather than creating new ones.
- When writing code, do not write newlines as the string "$\backslash n$", instead use the newline character.

Additional notes

- **Compute**: You have access to a $\{type_of_processor\}$ with the appropriate drivers installed.
- **Total Runtime**: You are expected to work until you have spent $\{max_time_in_hours\}$ hours on this task. You may iteratively develop your solution within these constraints, but bear in mind that program runtime also counts towards the time limit. This is a difficult task and we expect a good solution to require using the full time limit.
- **API keys**: You have been granted API keys for services necessary for the task. The API keys are available in the '.env' file at the root of your workspace. You can assume that these same API keys will already be set as environment variables in the reproduction environment when we evaluate your submission, so you do not need to set or export them in your submission.
- You have root access in your environment.
- You can use git for version control, avoid tracking large artefacts like datasets and model checkpoints. You can maintain a .gitignore for such formats, if required.
- Be mindful of running grep commands that scan through all files in the directory as it might take too long to execute.
- $\{multiple_hypotheses_line\}$You must use all the time available to you. Attempt to have multiple results for your proposed methods before the time runs out.
- You are also encouraged to note down your ideas, overall progress, analysis, limitations, etc. to keep track of your previous actions, as this is a long running research task, you should ensure to take notes of important steps that you might need for reference later
- Remember, you must actually write code, and make sure make changes to config files to actually execute the changes.