ContextEvolve: Multi-Agent Context Compression for Systems Code Optimization

Hongyuan Su [1 2] **Yu Zheng** [3] **Yong Li** [1 2]

Abstract

Large language models are transforming systems research by automating the discovery of performance-critical algorithms for computer systems. Despite plausible codes generated by LLMs, producing solutions that meet the stringent correctness and performance requirements of systems demands iterative optimization. Test-time reinforcement learning offers high search efficiency but requires parameter updates infeasible under API-only access, while existing training-free evolutionary methods suffer from inefficient context utilization and undirected search. We introduce **ContextEvolve**, a multi-agent framework that achieves RL-level search efficiency under strict parameter-blind constraints by decomposing optimization context into three orthogonal dimensions: a *Summarizer Agent* condenses semantic state via code-to-language abstraction, a *Navigator Agent* distills optimization direction from trajectory analysis, and a *Sampler Agent* curates experience distribution through prioritized exemplar retrieval. This orchestration forms a functional isomorphism with RL—mapping to state representation, policy gradient, and experience replay—enabling principled optimization in a textual latent space. On the ADRS benchmark, ContextEvolve outperforms state-of-the-art baselines by **33.3%** while reducing token consumption by **29.0%**. Codes for our work are released at https://anonymous.4open.science/r/ContextEvolve-ACC.

1. Introduction

Systems research has traditionally relied on human experts to design algorithms that optimize performance under strict constraints (Zoph & Le, 2016; Jiang et al., 2024a). The emergence of Large Language Models (LLMs) has enabled *AI-Driven Research for Systems* (ADRS) (Cheng et al., 2025; Jiang et al., 2024c), where LLMs automate the design of solutions for databases, networking, and distributed systems (Liu et al., 2025; Yu et al., 2025; Wooders et al., 2024). While LLMs can generate plausible code candidates, the stringent correctness and performance requirements of systems demand *rigorous iterative refinement*. Consequently, research focus has shifted from one-shot generation toward autonomous multi-round optimization (Jiang et al., 2024d): iteratively improving solutions until they exceed human-engineered baselines or exhaust computational budgets.

Test-time reinforcement learning (RL) offers a natural framework for such optimization via an RL loop of solution generation, reward evaluation, and LLM parameter update (Hubert et al., 2025). However, despite its promising performance, the scale of modern LLMs makes parameter updates computationally prohibitive (Kaplan et al., 2020). Meanwhile, leading LLM providers typically offer exclusively API-based access, precluding weight entirely. Training-free alternatives—evolutionary strategies such as AlphaEvolve (Novikov et al., 2025) and multi-agent frameworks such as CAMEL (Li et al., 2023)—sidestep this constraint but exhibit *low search efficiency*: they lack mechanisms for compressing accumulated context during iteration and for extracting precise optimization signals from long evolutionary history.

In this paper, we introduce **ContextEvolve**, a multi-agent framework designed for high search efficiency under parameter-blind constraints. Our key insight is that, instead of relying on a single monolithic model to manage the entire search state, optimization context can be decomposed into three orthogonal dimensions, each managed by a specialized agent. In ContextEvolve, a *Summarizer Agent* condenses semantic state, distilling code artifacts into natural language abstracts preserving critical state information; a *Navigator Agent* distills optimization direction, extracting textual gradients from trajectory analysis; and a *Sampler Agent* manages experience distribution, retrieving diverse, high-value exemplars as few-shot references. Crucially, this orchestration establishes a *functional isomorphism* with test-time

[1]Tsinghua University, Beijing, China [2]Zhongguancun Academy, Beijing, China [3]Massachusetts Institute of Technology, Cambridge MA, USA.

Preprint. January 29, 2026.

RL: the Summarizer Agent corresponds to state representation learning, the Navigator Agent emulates policy gradient estimation, and the Sampler Agent implements prioritized experience replay. This alignment enables ContextEvolve to inherit RL's sample efficiency while operating entirely in a text space without parameter access.

Our main contributions are:

- We propose ContextEvolve, a multi-agent framework achieving high search efficiency for system code optimization under API-only constraints via *structured context compression*.
- We introduce a suite of three specialized agents, including Summary, Gradient, and Sampler, which decompose context into semantic state, optimization direction, and experience distribution, and collectively approximate a test-time RL loop in a parameter-blind and textual space.
- On the ADRS benchmark across diverse systems code optimization domains, ContextEvolve surpasses state-of-the-art methods by **33.3%** while reducing token consumption by **29.0%**.

2. Related Works

Context Management. Long-horizon agentic workflows quickly accumulate large search state information, making context management a central bottleneck. Prior work addresses it from several angles: scaling model-side context capacity despite common failures such as positional bias and the lost-in-the-middle effect (Dai et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020; Borgeaud et al., 2022; Liu et al., 2024a; Zhang et al., 2024), externalizing long-term state into system-side memory (Lewis et al., 2020; Park et al., 2023; Packer et al., 2023), increasing information density via prompt compression and token pruning (Jiang et al., 2023; 2024b; Pan et al., 2024), and reducing prompt burden through agentic structuring that decomposes reasoning, search, and tool interaction into explicit loops (Yao et al., 2022; 2023; Shinn et al., 2023; Yang et al., 2024). However, these general approaches do not explicitly separate the distinct requirements of evolutionary systems code optimization, including preserving functional invariants, extracting improvement direction from noisy multi-metric trajectories, and maintaining diversity. In contrast, ContextEvolve decomposes context into semantic state, optimization direction, and experience distribution and assigns each to a specialized agent for high information density and low token cost across long runs.

LLM-based Evolutionary. For high-stakes code generation, one-shot LLM outputs often exhibit incorrectness or underperform test-time search loops that repeatedly propose candidates, evaluate with automated verifier, and update future proposals from historical feedback. Recent work instantiates this with evolutionary coding agents that treat programs as individuals and use evaluation-driven selection and mutation for open-ended evolution and algorithm discovery (Lange et al., 2025; Assumpção et al., 2025; Wan et al., 2025; Wu et al., 2024; Liu et al., 2024b). Closely related, prompt/instruction evolution optimizes discrete textual artifacts via population-based search and reflective mutation/selection (Fernando et al., 2023; Guo et al., 2023; Zhou et al., 2022; Pryzant et al., 2023; Yang et al., 2023); orthogonally, several approaches pursue more *directed* improvement signals without weight updates by storing verbal feedback or optimizing against model-provided critiques (Shinn et al., 2023; Yuksekgonul et al., 2025; Zhang et al., 2025). Nevertheless, existing evolutionary pipelines remain token-inefficient because they inflate prompts with raw history or compress context without explicit semantic disentanglement, leading to slower iteration and higher cost. Our proposed ContextEvolve targets these bottlenecks by maintaining compact semantic state, distilling textual gradients from weighted trajectory rollouts, and sampling exemplars via an RL-inspired experience distribution mechanism.

3. Preliminaries

3.1. Single-shot Generation

Let $\mathcal{C}$ denote the potentially discrete space of valid executable code solutions for a given problem. Given a task description $\mathcal{D}$, a trainable LLM $\mathcal{M}_\theta$ produces executable code $c \in \mathcal{C}$ via the sampling process $c \sim \mathcal{M}_\theta(\cdot \mid \mathcal{D})$. Then an automated evaluation oracle $\mathcal{E} : \mathcal{C} \to \mathbb{R}$ evaluates the code and returns a scalar score $s = \mathcal{E}(c)$ reflecting the solution quality.

3.2. Evolutionary Optimization

Training-free evolutionary methods treat the model parameters θ as immutable, and extend the single-shot paradigm to an iterative evolutionary process where the system iteratively refines solutions based on historical feedback. Define the optimization history $\mathcal{H}_t$ at step t as the sequence of generated solutions and their scores up to the previous step,

$$\mathcal{H}_t = \{(c_0, s_0), (c_1, s_1), \ldots, (c_{t-1}, s_{t-1})\}, \tag{1}$$

where each $c_i \in \mathcal{C}$ and $s_i = \mathcal{E}(c_i)$, and a context compression operator Φ that distills $\mathcal{H}_t$ into a short textual representation. The generation process at step t could be thus formulated as,

$$c_t \sim \mathcal{M}_\theta(\cdot \mid \Phi(\mathcal{H}_t), \mathcal{D}), \quad t \le T, \tag{2}$$

where T is the maximum iteration.

Our goal is to design the context compression operator Φ to maximize the expected score of the best solution,

$$\text{Maximize } \mathbb{E}\left[\max_{i=0}^{T} \mathcal{E}(c_i) \mid \Phi, \mathcal{D}\right]. \tag{3}$$

4. Method

4.1. Overall Framework

We propose ContextEvolve, a multi-agent framework for ADRS that compresses raw interaction logs into a refined optimization context, thereby enabling high search efficiency under strict parameter-blind constraints. To effectively compress the extensive evolutionary experience into a finite window, our framework decomposes the optimization context into three orthogonal dimensions: *semantic state*, *optimization direction*, and *experience distribution*. Accordingly, we introduce three specialized agents to distinctively manage them. Specifically, the Summarizer Agent maintains the *semantic state* by condensing complex code artifacts into concise natural language abstracts. The Navigator Agent steers the *optimization direction* by distilling promising search guidance from historical trajectories. Finally, the Sampler Agent modulates the *experience distribution* by curating diverse and high-value exemplars to serve as instructive few-shot references. This decomposition and agent specialization allow our approach to construct a compact, semantically rich context representation that effectively utilizes previous experience for the next generation.

Beyond context compression, we also establish a functional isomorphism with RL algorithms, where our agents collaboratively approximate key mechanisms of search efficiency purely in the natural language space without any parameter updates. Specifically, the Summarizer Agent corresponds to state representation learning, the Navigator Agent emulates policy gradient updates, and the Sampler Agent implements prioritized experience replay. This theoretical alignment ensures that our framework is not merely a heuristic search but a principled isomorphism of RL, thereby significantly highlighting the search efficiency of ContextEvolve.

The collaborative workflow of these agents constitutes our evolutionary pipeline, as detailed in Algorithm 1. In each iteration, the process begins by selecting a parent solution c^p from the Evolve Buffer based on pre-defined criteria. First, the Navigator Agent analyzes multiple evolution trajectories rolled out from the Evolve Buffer. By scrutinizing the correlation between code modifications and metric fluctuations, it distills optimization directions for potential performance gains. Next, conditioned on the parent's semantic state and this directional guidance, the Sampler Agent curates a set of instructive exemplars from the population to serve as few-shot references. Subsequently, the Generator Agent integrates the parent, the directional guidance, and the retrieved exemplars to generate a superior offspring c^c, which is immediately assessed by the evaluator for its fitness. Following evaluation, the Summarizer Agent compares the offspring against its parent to abstract the key characteristics into a new semantic summary. Finally, the new generated code along with its abstract and fitness will be added to the Evolve Buffer and ready for the next iteration.

Algorithm 1 ContextEvolve: Multi-Agent Evolutionary Pipeline (with RL Functional Isomorphism)

Input: Task $\mathcal{D}$, LLM $\mathcal{M}_\theta$, Evaluator $\mathcal{E}$, Ancestor c_0
Input: Env $\mathcal{E}_{env}$, Policy π_θ, Reward $\mathcal{R}$, Initial State $\mathbf{s}_0$
Initialize: Evolve Buffer $\mathcal{B}_e \leftarrow \{(c_0, s_0, z_0)\}$
Initialize: Replay Buffer $\mathcal{B}_r \leftarrow \emptyset$
for $t = 1$ to T **do**
  // Phase 1: Semantic State Selection
  $(c_t^p, z_t^p, s_t^p) \leftarrow \text{SelectParent}(\mathcal{B}_{evolve})$
  $\mathbf{s}_t \leftarrow \text{Observe}(\mathcal{E}_{env}); \quad e_t = \text{Encoder}(\mathbf{s}_t)$
  // Phase 2: Optimization Direction
  $\tau \leftarrow \text{Rollout}(\mathcal{B}_e); \quad g_t \leftarrow \text{GradientAgent}(\tau)$
  $\tau' \sim \mathcal{B}_r; \quad \nabla J \leftarrow \text{Estimator}(\tau')$
  // Phase 3: Experience Distribution
  $E_{ctx} \leftarrow \text{SamplerAgent}(\mathcal{B}_e, z_t^p, g_t)$
  $\tau' \leftarrow \text{PrioritizedSample}(\mathcal{B}_r)$
  // Phase 4: Context Construction
  $\Phi_t \leftarrow \text{Compose}(z_t^p, g_t, E_{ctx})$
  $\Delta\theta \leftarrow \nabla_\theta \mathbb{E}_\tau[J(\tau')]$
  // Phase 5: Code Generation
  $c_t^c \sim \mathcal{M}_\theta(\cdot \mid \Phi_t)$
  $\theta' \leftarrow \theta + \alpha\Delta\theta; \quad a_t \sim \pi_{\theta'}(\cdot \mid \mathbf{s}_t)$
  // Phase 6: Evaluation
  $s_t^c \leftarrow \mathcal{E}(c_t^c)$
  $\mathbf{r}_t \leftarrow \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t)$
  // Phase 7: Semantic State
  $z_t^c \leftarrow \text{SummaryAgent}(z_t^p, c_t^c)$
  $\mathbf{s}_{t+1} \leftarrow \mathcal{E}_{env}(\mathbf{s}_t, \mathbf{a}_t); \quad e_{t+1} = \text{Encoder}(\mathbf{s}_{t+1})$
  // Phase 8: Buffer Updat
  $\mathcal{B}_e \leftarrow \mathcal{B}_e \cup \{(c_t^p, s_t^p, z_t^p; c_t^c, s_t^c, z_t^c)\}$
  $\mathcal{B}_r \leftarrow \mathcal{B}_r \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})\}$
end for
Output: Best solution c^* with the highest score s^*

4.2. Specialized Context Compression Agents

While training-free inference-time optimization demonstrates potential in ADRS, current approaches still face significant challenges regarding search efficiency and information density within limited contexts. Frameworks like OpenEvolve retain the original optimization history within a restricted window, which results in the underutilization of context, leading to search efficiency and suboptimal outcomes. This challenge underscores the critical need for effective context management in the face of accumulating information. To address this, we decompose the optimization context into three largely orthogonal semantic dimensions: *semantic state*, *optimization direction*, and *experience distribution*, and employ three specialized agents to distinctively maintain them.

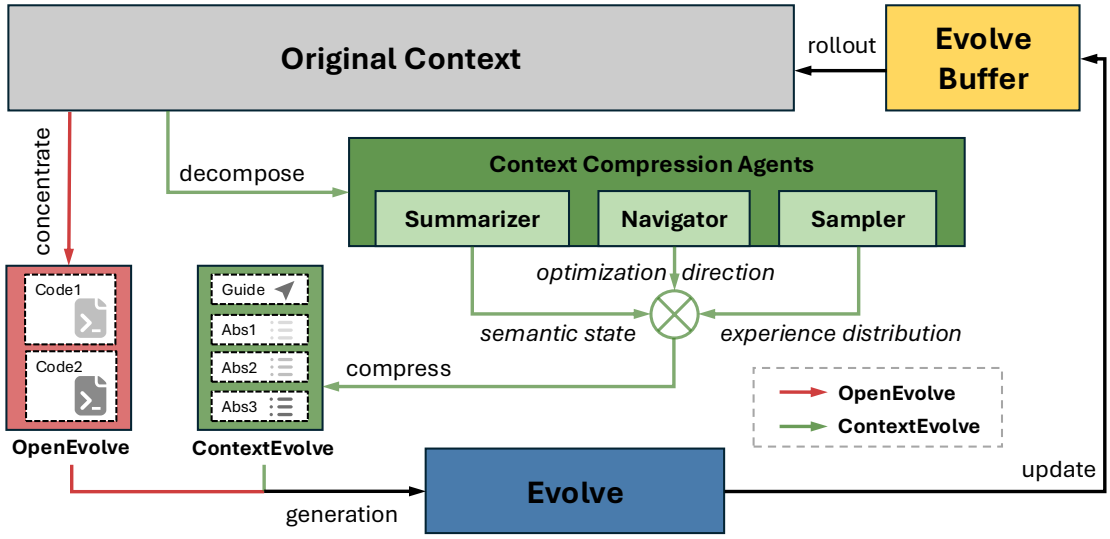

Figure 1. The pipeline comparison of ContextEvolve and OpenEvolve. OpenEvolve (red) directly concentrates the few original codes in limited context window, leading to low information density and inefficient evolutionary search. ContextEvolve (green) leverages specialized context distillation agents to compress the lengthy context, enriching the limited window with numerous valuable information.

Summarizer Agent: Semantic State Condensation

The Summarizer Agent maintains and condenses the *semantic state*. It encodes high-dimensional code into a concise, high-level textual abstract that strips away redundancy while preserving critical property. To ensure the abstract captures both the innovative designs of the offspring and the vital functional segments inherited from the parent, we provide the agent with the parent's abstract z_p and the offspring's raw code c_c. The Summarizer Agent is tasked with summarizing both novel aspects and preserved elements, formulated as

$$z^c \sim \mathcal{M}_{\theta_1}(\cdot \mid \text{Prompt}_{\text{summary}}(z^p, c^c)). \tag{4}$$

By transforming raw code differences and similarities into dense semantic descriptions, the Summarizer Agent packs more insights into the limited context window, ensuring that critical historical experience remains accessible throughout entire evolutionary process.

Navigator Agent: Optimization Direction Distillation

The Navigator Agent governs the *optimization direction* by distilling high-level guidance from historical successes and failures. To ensure the offspring benefits from high-quality past improvements, the agent samples trajectories based on metric variations $\Delta s = s_{parent} - s_{child}$. These trajectories are weighted and sampled based on Δs across three distinct categories: consistent improvement, mixed fluctuation, and consistent decline. By recording the evolution of abstracts and the corresponding metric shifts, the agent distills directional guidance,

$$\begin{aligned} g_t =& \text{GradientAgent}(\{\tau \sim p(\Delta s)_{j=1}^{m}\}) \\ & \sim \mathcal{M}_{\theta_2}(\cdot \mid \text{Prompt}_{\text{summary}}\{(z_i^p, z_i^c, \Delta s_i)\}_{i\in\tau}), \end{aligned} \tag{5}$$

where m is the number of sampled trajectories and $p(\Delta s)$ weights the categories.

By analyzing the correlation between algorithmic changes and metric fluctuations, the Navigator Agent prevents the evolution from repeated futile attempts and steers it toward unexplored high-potential regions, significantly accelerating convergence toward optimal solutions.

Sampler Agent: Experience Distribution Modulation

The Sampler Agent regulates the *experience distribution* by curating a small set of the most informative solutions to serve as quality few-shot exemplars. To balance exploration and exploitation, the agent selects individuals from the population based on their relevance, diversity, and proven outcomes, considering the parent state z^p and the current guidance g,

$$E_{ctx} \sim \mathcal{M}_{\theta_3}(\cdot \mid \text{Prompt}_{\text{sample}}(\mathcal{B}_e, z^p, g)). \tag{6}$$

By populating the context window with high-value demonstrations rather than random history, the Sampler Agent provides the generator with the most relevant references for the current optimization step while maximizing the utility of each token.

In conclusion, these specialized agents collaboratively transform context management from mere log buildup into an active compression process. This capability enables the system to conduct deep, long-horizon searches within a

fixed-length context window and achieve robust optimization performance with minimal token consumption.

4.3. Functional Isomorphism with RL

The efficiency of ContextEvolve comes from its active context compression. However, beyond this information-theoretic perspective, we observe a profound functional isomorphism between our multi-agent framework and the fundamental components of RL. Notably, this alignment emerges naturally from the logical decomposition of context management rather than guiding its initial creation. Given that RL frameworks are renowned for their high sample efficiency in complex decision spaces, this isomorphism provides strong support for the superior search capabilities and token efficiency of our parameter-blind framework. Specifically, just as RL agents maximize cumulative rewards by iteratively updating representations, directions, and experiences, our agents collaborate to refine solutions through analogous mechanisms.

SEMANTIC STATE AS STATE REPRESENTATION

In RL, raw observations are often high-dimensional and noisy. Effective learning relies on an encoder that condenses these observations into a compact latent feature vector, capturing the essential dynamics required for downstream decision-making. Similarly, the Summarizer Agent condenses high-dimensional code artifacts c into a concise semantic abstract z,

$$\text{SummaryAgent}(z^p, c^c) \Leftrightarrow \text{Encoder}(\mathbf{o}_t). \tag{7}$$

TEXTUAL GUIDANCE AS POLICY GRADIENT

Policy gradients in RL derive directional updates from sampled trajectories to maximize the expected return. In our setting, since the model parameters θ are frozen, the context prompt Φ serves as the effective adjustable parameter set. The Navigator Agent performs an operation analogous to gradient estimation by distilling improvement directions from weighted historical trajectories,

$$\text{GradientAgent}(\tau) \Leftrightarrow \nabla_\theta \mathbb{E}_\tau[J(\tau)]. \tag{8}$$

This isomorphism reveals that ContextEvolve performs gradient ascent in the semantic space, offering a directed search mechanism more efficient than random mutation.

CONTEXT SAMPLING AS PRIORITIZED EXPERIENCE

Off-policy RL gains massive efficiency by breaking temporal correlations and reusing past transitions via a Replay Buffer. The Sampler Agent implements a semantic version of this, retrieving exemplars E_{ctx} conditioned on the current semantic state and directional guidance:

$$\text{SamplerAgent}(\mathcal{B}_e, z^p, g) \Leftrightarrow \text{PS}(\mathcal{B}_r), \tag{9}$$

It enables the model learns from the most instructive historical failures and successes rather than the immediate past.

SYSTEM-LEVEL STRUCTURAL ALIGNMENT

Beyond the context agents, the structure of ContextEvolve mirrors the fundamental architecture of an RL system:

- **Generator Agent as Policy Network:** The generator $\mathcal{M}_\theta$, conditioned on the dynamic context Φ, functions as the stochastic policy $\pi(\cdot|\mathbf{s})$. By integrating the compressed state, direction, and exemplars, it executes the generative action that maps the current context to the solution space.
- **Evolve Buffer as Replay Buffer:** The storage of tuples (c, s, z) in $\mathcal{B}_{evolve}$ is functionally equivalent to the replay buffer $\mathcal{D} = \{(\mathbf{s}, a, r, \mathbf{s}')\}$. This decoupling of data generation from data utilization allows our agents to perform off-policy optimization, extracting global insights from the entire exploration history.

In conclusion, this isomorphism reveals that ContextEvolve is not merely a heuristic search method but a principled approximation of an RL system operating in a textual latent space. By reconstructing the mechanisms of state representation, gradient guidance, and experience replay using natural language agents, ContextEvolve inherits RL-like sample efficiency in a completely training-free setting.

5. Experiments

5.1. Experimental Setup

We evaluate ContextEvolve on five challenging scenarios from the ADRS benchmark: Transaction Scheduling (TS), SQL Optimization (SQL), Load Balancing (LB), Sparse Attention Kernel (SAK), and Model Placement (MP). Specific details about these tasks are provided in Appendix A.

We first compare ContextEvolve against static generation methods, including: (1) **Heuristics** (Cheng et al., 2025), representing traditional rule-based algorithms widely deployed in production systems; (2) **Human-SOTA** (Cheng et al., 2024; Liu et al., 2025; Yu et al., 2025; Desai et al., 2025), denoting state-of-the-art solutions manually crafted by domain experts; and (3) **LLM One-shot**, where the model produces a solution in a single pass based on the problem description, serving as the performance lower bound for LLM-based capabilities. We also compare against advanced LLM-based evolutionary approaches, including (4)

Table 1. Overall performance comparison on ADRS benchmark. We report the key domain-specific metrics along with combined score.

Method	**TS** (100 iters)			**SQL** (100 iters)			**LB** (300 iters)			**SAK** (100 iters)			**MP** (100 iters)		
	Make.↓	Corr.↑	Score↑	Hit.↑	Lat.↓	Score↑	Bal.↑	Spe.↑	Score↑	Dens.↓	Err.↓	Score↓	Press.↑	Succ.↑	Score↑
Heuristics	38.50	1.00	25.91	0.56	51.86	0.53	0.25	0.20	0.13	0.731	0.481	0.606	20.89	1.00	21.89
Human-SOTA	32.40	1.00	30.80	0.69	17.43	0.66	0.24	0.43	0.14	0.717	0.469	0.593	21.34	1.00	22.34
LLM One-shot	35.80	1.00	27.86	0.64	0.69	0.66	0.25	0.20	0.13	0.730	0.471	0.600	21.03	0.96	21.99
GEPA	29.00	1.00	34.36	0.72	0.65	0.73	0.25	0.20	0.14	**0.627**	0.575	0.602	21.49	1.00	22.49
OpenEvolve	29.70	1.00	33.56	0.71	2.01	0.72	0.25	0.45	0.15	0.727	**0.454**	0.591	21.67	1.00	22.67
ContextEvolve	**27.60**	1.00	**36.10**	**0.78**	**0.56**	**0.79**	**0.34**	**0.65**	**0.20**	0.676	0.496	**0.586**	**23.02**	1.00	**24.02**
Impr.%	-4.8	+0.0	+5.1	+8.3	-13.9	+8.2	+36.0	+44.4	+33.3	-7.8	-9.2	+0.9	+6.2	+0.0	+6.0

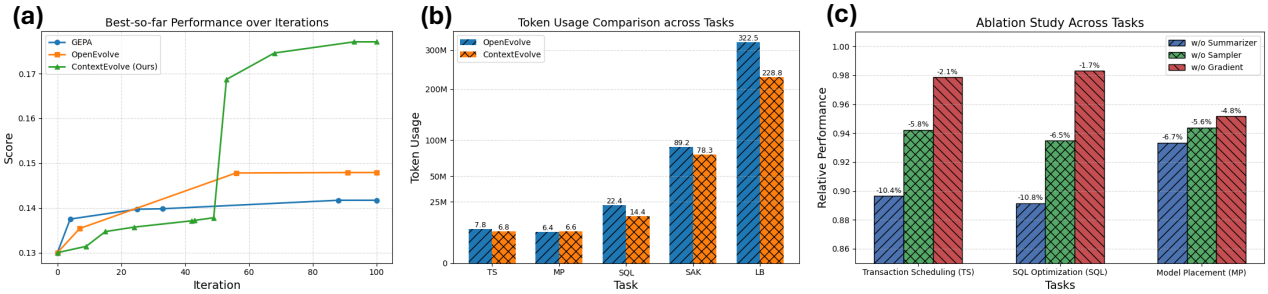

Figure 2. Efficiency analysis and ablation studies of ContextEvolve. **(a)** Best-so-far performance trajectories over evolutionary iterations in the LB task. **(b)** Cumulative token usage across five ADRS tasks. **(c)** Relative performance of ablated variants.

OpenEvolve (an open-source implementation of AlphaEvolve) (Sharma, 2025), and (5) **GEPA** (Agrawal et al., 2025), a prompt optimization framework that employs reflective mutation and Pareto-based selection. Since these tasks universally require optimizing trade-offs, we report two competing metrics as well as a weighted combined score which calculated based on the domain-specific importance. We utilize Qwen3 (Yang et al., 2025) as the underlying foundation model for all LLM-based components, including the context agents in our framework and the baseline generators.

5.2. Overall Performance

The comparison results on the ADRS benchmark between ContextEvolve and baselines are presented in Table 1 where we can derive the following observations:

- **Existing baselines yield sub-optimal solutions due to search inefficiencies.** Static generation methods, including Heuristics, Human-SOTA, and LLM One-shot, generally establish the lower bound for performance. Especially in the TS task, the Human-SOTA solution lags behind the evolutionary method by over 17%. These methods rely on pre-defined rules or single-pass inference and lack the feedback loops necessary to navigate high-dimensional search spaces. Though evolutinary methods like GEPA and OpenEvolve improve upon static baselines by leveraging iterative feedback, they suffer from inefficient context utilization, limiting the depth of exploration within a fixed budget. On average, they always obtain sub-optimal solution and trail ContextEvolve by 6.5% in overall scores.
- **ContextEvolve displays significant advantages through multi-agent context compression.** Our approach leverages multi-agent collaboration and orthogonal context decomposition to actively compress optimization histories, addressing the challenges of unguided search and information bloat. Through specialized agents for context compression, it comprehensively surpasses existing baselines across all five ADRS tasks, achieving an average score improvement of 6.5% over the top baseline. Particularly in the LB task, where baselines struggle to optimize the balance metric and only yield minor gains in speed, our method elevates balance by over 36% while delivering the fastest speed. These results underscore the necessity and effectiveness of context compression in evolutionary optimization, enabling outstanding search efficiency with superior token utility.

We further analyze the efficiency of our framework by examining both the evolutionary trajectory and the token cost associated with the search process. As illustrated in Figure 2(a), evolutionary baselines converge rapidly within the first 60 iterations and expend over 40% of total attempts on failed or repetitive exploration to achieve less than a 0.1% score improvement, indicating a low marginal utility of search. In contrast, ContextEvolve maintains a continuous upward trajectory by context compression, which updates the best-so-far solution 83.3% more frequently than baselines. Notably, when baselines stagnate in local optima, ContextEvolve achieves a 22.4% score breakthrough and sustain steady improvements.

To validate our context compression strategy, Figure 2(b)

compares cumulative token usage against OpenEvolve. A counter-intuitive finding is that while our multi-agent framework requires over $3\times$ more API calls, total token consumption is universally lower, averaging a 17.3% reduction. Whereas OpenEvolve concatenates multiple raw codes directly into prompts, ContextEvolve curates fewer high-value exemplars based on condensed semantic states, significantly boosting information density per token. Furthermore, this efficiency gain scales with task complexity. For lightweight tasks like Transaction Scheduling (TS) and Model Placement (MP) with short code solutions, the consumption remains comparable (difference $< 5\%$). But for complex tasks like Load Balancing (LB) that involve lengthy implementations, our method saves nearly 30%, proving its strong handling of high-context loads.

5.3. Ablation Study

We conduct an ablation study to validate the efficacy of our multi-agent architecture. We evaluate variants of ContextEvolve by independently removing each context management agent and illustrate the results in Figure 2(c). The removal of any agent leads to a decline in final solution performance, with average impacts ranked as Summarizer Agent (-9.3%), Sampler Agent (-6.0%), and Navigator Agent (-2.9%). These results confirm that our architecture effectively manages distinct context dimensions, and the absence of any agent results in a loss of critical context information, thereby reducing search efficiency and overall performance degradation. Notably, removing the Summarizer Agent yields the most severe regression, pushing performance close to the LLM one-shot baseline. Specifically, the scores on TS, SQL, and MP deteriorate by 10.4%, 10.8%, and 6.7%, respectively. Without such semantic condensation, the limited context window becomes dominated by verbose syntax and low-level details, sharply reducing information density and directly impairs the reasoning capabilities of the Generator Agent. Furthermore, the absence of concise summaries hinders the Gradient and Sampler Agents from distilling valid optimization directions and curating relevant exemplars, which indirectly destabilizes the overall evolutionary performance through cascading effects in agent cooperation.

5.4. Case Study: Load Balancing Task

We conduct a case study on the LB task to dissect how ContextEvolve achieves algorithmic breakthroughs compared to evolutionary baselines. The evolution begins with a standard greedy implementation derived from DeepSeek (DeepSeek AI, 2024). After 10 evolutionary iterations, both OpenEvolve and ContextEvolve independently discover a "Snake Round-Robin" heuristic strategy. This approach replaces the inefficient linear search in the packing phase with a pre-calculated zigzag assignment pattern, significantly improving assignment speed. Notably, this

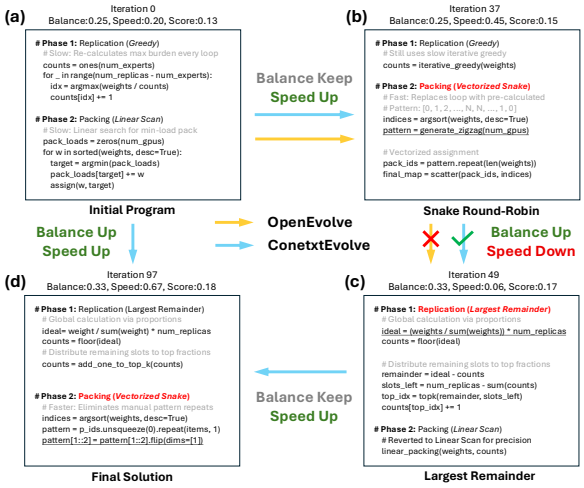

Figure 3. **(a)** The initial solution adopts greedy replication and linear-scan packing, achieving moderate balance but limited speed. **(b)** A vectorized snake round-robin assignment improves runtime while preserving balance. **(c)** Largest-remainder proportional apportionment yields balance gains at the cost of reduced speed. **(d)** The final solution recovers speed without sacrificing balance.

variant is not publicly available (Cheng et al., 2025) and is absent from the base model's training data (Yang et al., 2025), highlighting the capability of LLM-based evolution to discover advanced algorithmic concepts.

Afterward, the optimization trajectories diverge. While OpenEvolve generates only minor syntactic variations of the zigzag pattern until the budget is exhausted, ContextEvolve analyzes historical bottlenecks and identifies that heuristic sorting alone fails to handle extreme tail loads. As illustrated in Figure 3(c), by the 49th iteration ContextEvolve proposes a fundamental algorithmic shift in the replication stage, abandoning the iterative greedy approach in favor of a largest remainder proportional apportionment strategy. Specifically, it calculates the ideal replica count based on the expert's global load proportion, floors this value to ensure minimum allocation, and prioritizes distributing the remaining capacity to experts with the largest fractional remainders. This rigorous global allocation boosts the balancedness score from 0.25 to 0.33 (+32%). Despite the gain in balance, this sophisticated allocation introduces computational overhead via complex tensor operations, initially causing the speed score to drop. To address this, ContextEvolve retrieves both the previous high-speed solution and the new high-balance solution as in-context exemplars. Leveraging these references, the framework re-integrates the searched "Snake Round-Robin" strategy, and further introduces closed-form tensor transformations (see Figure 3(d)) to substantially strengthen the speed advantage. This optimization restores and exceeds the original speed score while maintaining superior balancedness, outperforming the baselines on both metrics simultaneously.

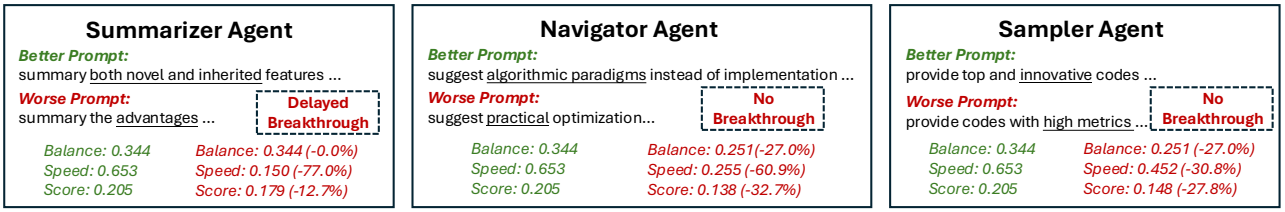

Figure 4. Takeaways from prompt perturbations. **(a)** Preserving ancestral traits is as critical as capturing innovation. **(b)** Directional ambiguity guidance outperforms implementation specificity. **(c)** Sampling should prioritize informative semantics, not only high scores.

5.5. Design Takeaways

Beyond demonstrating performance superiority, we aim to distill high-level design concepts for context management that may generalize to other LLM-based inference-time search methods. Unlike the structural ablation study, we maintain the macro-level multi-agent architecture but apply specific micro-level perturbations to the prompt design of each agent. We visualize these prompt modifications and their performance on the Load Balancing (LB) task in Figure 4, from which we derive three critical insights.

Preserving ancestral traits is as critical as capturing innovation. For the Summarizer Agent, effective semantic condensation must balance the extraction of novel modifications with the retention of inherited features. When we modify the prompt to summarize only the novel changes of an offspring, the performance drops by more than 12.6%. Further analysis of the evolutionary trajectory further shows that while the key heuristic strategy (see Section 5.4 (c)) still emerges, its discovery is delayed substantially from the 49th to the 87th iteration. The underlying cause is an amnesia effect: beneficial inherited characteristics are present in the raw code but omitted in the semantic state. As a result, other agents that rely on summaries repeatedly misinterpret well-explored directions as underexplored opportunities, leading to redundant trials and degraded search efficiency.

Directional ambiguity guidance outperforms implementation specificity. For the Navigator Agent, ambiguity in high-level direction is beneficial, while over-specification is detrimental. We modified the gradient prompt to provide specific, practical implementation steps rather than abstract directions, performance significantly drops to 0.138 (-32.7%), failing to find advanced heuristics. This collapse arises because highly specific instructions implicitly commit the search to a narrow set of edits before candidate generation. Consequently, the Generator Agent is forced to instantiate a prescribed plan with limited degrees of freedom, behaving more like a code formatter than an explorer of alternative algorithmic branches. This prematurely narrows the effective solution space, suppresses generative diversity, and makes the optimization trajectory prone to local optima, thereby hindering the discovery of new paradigms.

Sampling should prioritize informative semantics, not only high scores. For the Sampler Agent, high-scoring exemplars are not the sole source of valuable information, as low-quality or even failed individuals often contain the seeds of breakthroughs. When we restrict the sampler to return only the highest-scoring individuals, the final score decreases to 0.148 (-27.8%), only on par with OpenEvolve, and the run fails to identify advanced heuristics. A retrospective analysis shows that the decisive inspiration for the final breakthrough on balancedness in the original ContextEvolve comes from a failed "vectorized binary-search thresholding strategy," which received a score of 0 due to implementation errors. Although functionally broken, its underlying logic of heuristic allocation was semantically innovative, motivating the generation of the best-performing "Largest Remainder" strategy. Through semantic state analysis, the Sampler Agent should carefully extract promising concepts even from low-scoring individuals, allowing the system to iterate on flawed but brilliant ideas and preventing the premature discard of potentially transformative insights due to imperfect preliminary implementations.

6. Discussion

In this work, we present ContextEvolve, a multi-agent framework that addresses the inefficiency of training-free evolutionary search for systems code optimization. By decomposing the optimization history into orthogonal dimensions, our approach effectively overcomes the information bottlenecks. This architecture establishes a functional isomorphism with reinforcement learning, indicating its high search efficiency within parameter-blind setting. Empirical results on the ADRS benchmark demonstrate that ContextEvolve significantly outperforms state-of-the-art baselines in solution quality while substantially reducing token consumption.

Despite these advancements, several avenues remain for future investigation. Our research focuses primarily on functionally isolated algorithm. Scaling ContextEvolve to optimize large-scale codebases with complex inter-module dependencies requires further study. Moreover, developing stabilizing mechanisms to mitigate the high variance exacerbated by LLM is critical for ensuring consistent outcomes. Finally, we will investigate evaluation mechanisms to promote the discovery of diversity algorithmic diverse algorithmic paradigms.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., Opsahl-Ong, K., Singhvi, A., Shandilya, H., Ryan, M. J., Jiang, M., et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Assumpção, H., Ferreira, D., Campos, L., and Murai, F. Codeevolve: An open source evolutionary coding agent for algorithm discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., Van Den Driessche, G. B., Lespiau, J.-B., Damoc, B., Clark, A., et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Cheng, A., Kabcenell, A., Chan, J., Shi, X., Bailis, P., Crooks, N., and Stoica, I. Towards optimal transaction scheduling. *Proceedings of the VLDB Endowment*, 17 (11):2694–2707, 2024.

Cheng, A., Liu, S., Pan, M., Li, Z., Wang, B., Krentsel, A., Xia, T., Cemri, M., Park, J., Yang, S., Chen, J., Agrawal, L., Desai, A., Xing, J., Sen, K., Zaharia, M., and Stoica, I. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2978–2988, 2019.

DeepSeek AI. EPLB: Expert parallelism load balancer. https://github.com/deepseek-ai/EPLB, 2024. GitHub repository, accessed January 2026.

Desai, A., Agrawal, K. K., Yang, S., Cuadron, A., Schroeder, L. G., Zaharia, M., Gonzalez, J. E., and Stoica, I. vattention: Verified sparse attention. *arXiv preprint arXiv:2510.05688*, 2025.

Fernando, C., Banarse, D., Michalewski, H., Osindero, S., and Rocktäschel, T. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., and Yang, Y. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Hubert, T., Mehta, R., Sartran, L., Horváth, M. Z., Žužić, G., Wieser, E., Huang, A., Schrittwieser, J., Schroecker, Y., Masoom, H., et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pp. 1–3, 2025.

Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., and Yu, Y. Llmopt: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*, 2024a.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, 2024b.

Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, 2024c.

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

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Lange, R. T., Imajuku, Y., and Cetin, E. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Liu, S., Chen, C., Qu, X., Tang, K., and Ong, Y.-S. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2024b.

Liu, S., Ponnapalli, S., Shankar, S., Zeighami, S., Zhu, A., Agarwal, S., Chen, R., Suwito, S., Yuan, S., Stoica, I., et al. Supporting our ai overlords: Redesigning data systems to be agent-first. *arXiv preprint arXiv:2509.00997*, 2025.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Packer, C., Fang, V., Patil, S., Lin, K., Wooders, S., and Gonzalez, J. Memgpt: Towards llms as operating systems. 2023.

Pan, Z., Wu, Q., Jiang, H., Xia, M., Luo, X., Zhang, J., Lin, Q., Rühle, V., Yang, Y., Lin, C.-Y., et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*, 2024.

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Pryzant, R., Iter, D., Li, J., Lee, Y. T., Zhu, C., and Zeng, M. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Sharma, A. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/codelion/openevolve.

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

Wan, C., Dai, X., Wang, Z., Li, M., Wang, Y., Mao, Y., Lan, Y., and Xiao, Z. Loongflow: Directed evolutionary search via a cognitive plan-execute-summarize paradigm. *arXiv preprint arXiv:2512.24077*, 2025.

Wooders, S., Liu, S., Jain, P., Mo, X., Gonzalez, J. E., Liu, V., and Stoica, I. Cloudcast:{High-Throughput},{Cost-Aware} overlay multicast in the cloud. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 281–296, 2024.

Wu, X., Wu, S.-h., Wu, J., Feng, L., and Tan, K. C. Evolutionary computation in the era of large language model: Survey and roadmap. *IEEE Transactions on Evolutionary Computation*, 2024.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

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

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yu, S., Xing, J., Qiao, Y., Ma, M., Li, Y., Wang, Y., Yang, S., Xie, Z., Cao, S., Bao, K., et al. Prism: Unleashing gpu sharing for cost-efficient multi-llm serving. *arXiv preprint arXiv:2505.04021*, 2025.

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Lu, P., Huang, Z., Guestrin, C., and Zou, J. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zhang, Y., Zhang, Y., Leach, K., and Huang, Y. Codegrad: Integrating multi-step verification with gradient-based llm refinement. *arXiv preprint arXiv:2508.10059*, 2025.

Zhang, Z., Chen, R., Liu, S., Yao, Z., Ruwase, O., Chen, B., Wu, X., Wang, Z., et al. Found in the middle: How language models use long contexts better via plug-and-play positional encoding. *Advances in Neural Information Processing Systems*, 37:60755–60775, 2024.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. Steering large language models using ape. In *NeurIPS ML Safety Workshop*, 2022.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A. ADRS Benchmark

The ADRS benchmark contains five challenging optimization tasks that demonstrate the versatility and effectiveness of ContextEvolve in different contexts. Below, each task is described along with its specific objectives and the trade-offs involved in optimizing the solution.

- Transaction Scheduling (TS): This task involves optimizing the execution order of transactions in a database system to minimize the total execution time, also known as the makespan (Make.). The optimization problem requires balancing the trade-off between transaction order and latency. The goal is to maximize the correctness rate (Corr.) and throughput.
- SQL Optimization (SQL): In SQL Optimization, the objective is to reorder rows and fields of a table to maximize the hit rate in a Key-Value (KV) cache, which ultimately improves the speed of SQL query inference. The challenge lies in balancing the prefix cache hit rate (Hit.) against the latency (Lat.) of the reordering algorithm.
- Load Balancing (LB): The Load Balancing task focuses on optimizing the distribution of computational load across multiple GPUs, such as in a Mixture of Experts (MoE) model. The key objective is to maximize the load imbalance (Bal.), ensuring that each GPU handles a proportionate share of the computational workload while also maximizing the speed (Spe.) of rebalancing the load across GPUs when changes occur.
- Sparse Attention Kernel (SAK): This task aims to optimize sparse attention mechanisms in neural network models, where the goal is to strike a balance between the density of the active indices in the attention mask and the relative error introduced by this sparsity. The optimization involves designing attention masks that minimize the combined loss of density (Dens.) and relative error (Err.).
- Model Placement (MP): The Model Placement task involves optimizing the placement of large machine learning models across multiple GPUs to reduce contention and ensure efficient resource utilization. The optimization focuses on minimizing the KV pressure ratio (KVPR) across GPUs, which is a measure of cache contention. The task seeks to reduce the KVPR (higher Press.) to improve the performance of model inference while ensuring successful (Succ.) utilization of GPUs.

B. Best Evolved Code

In this section, we present specific examples of the highest-performing code solutions discovered by ContextEvolve during the evolutionary process. Specifically, Figure 5 provides a side-by-side comparison of the evolutionary breakthrough in the Load Balancing (LB) task. The initial solution relies on iterative Python loops, while the best solution discovered by ContextEvolve utilizes vectorized tensor operations and a rigorous proportional allocation strategy. The initial codes and the best evolved code for all the five tasks are provided in our code repository. These artifacts demonstrate the ability of our framework to discover complex algorithmic logic, such as proportional apportionment and heap-based optimization, which are absent in the initial seed code.

(a) Initial Code

Characteristics: Iterative Greedy, Linear Scan, Python Loops

```python
def balanced_packing(weight, num_packs):

    Greedy Linear Scan Strategy
    Complexity: O(Num_Layers * Num_Groups)

    num_layers, num_groups = weight.shape
    groups_per_pack = num_groups // num_packs

    # ... setup code omitted ...

    # SLOW: Python-level nested loops
    for i in range(num_layers):
        pack_weights = [0] * num_packs
        pack_items = [0] * num_packs

        # Iterating over every group
        for group in indices[i]:
            # Linear scan to find min-load pack
            pack = min(
                (k for k in range(num_packs)
                 if pack_items[k] < groups_per_pack),
                key=pack_weights.__getitem__,
            )

            # Scalar assignment (High Overhead)
            pack_index[i, group] = pack
            pack_weights[pack] += weight[i, group]

    return pack_index, rank_in_pack

def replicate_experts(weight, num_phy):

    Iterative Greedy Replication
    Complexity: O(Num_Replicas * Num_Layers)

    n, num_log = weight.shape
    # ... setup code omitted ...

    logcnt = torch.ones(...)

    # SLOW: Loop once per extra replica needed
    for i in range(num_log, num_phy):
        # Recalculate max load at every step
        scores = weight / logcnt
        idx = scores.max(dim=-1).indices

        # Incremental update
        phy2log[:, i] = idx
        rank[:, i] = logcnt[..., idx]
        logcnt[..., idx] += 1

    return phy2log, rank, logcnt
```

(b) Best Evolved Code (ContextEvolve)

Characteristics: Snake Heuristic, Proportional, Vectorized

```python
def balanced_packing(weight, num_packs):

    Vectorized Snake Round-Robin Strategy
    Complexity: O(1) Tensor Operations

    num_layers, num_items = weight.shape
    device = weight.device

    # Sort weights descending
    sorted_indices = torch.argsort(weight, descending=
     True)

    # INNOVATION 1:  Snake Pattern Generation
    # Pattern: 0, 1, 2, ..., N, N, ..., 1, 0
    p_ids = torch.arange(num_packs, device=device)
    pattern = p_ids.unsqueeze(0).repeat(items_per_pack,
      1)
    # Flip every second row to create zig-zag
    pattern[1::2] = pattern[1::2].flip(dims=[1])

    # Flatten to match items
    pack_ids = pattern.flatten()[:num_items]

    # INNOVATION 2:  Vectorized Assignment
    # Eliminates Python loops entirely
    pack_index.scatter_(1, sorted_indices, pack_ids)

    return pack_index, rank_in_pack

def replicate_experts(weight, num_phy):

    Proportional Apportionment (Largest Remainder)
    Complexity: O(1) via Sort/TopK

    num_layers, num_log = weight.shape

    # INNOVATION 3:  Global Calculation
    # Calculate ideal share based on weight ratio
    total = weight.sum(dim=1, keepdim=True)
    ideal = (weight * num_phy) / total

    # Assign integer parts immediately
    logcnt = ideal.floor().to(torch.int64)

    # INNOVATION 4:  Deficit Filling
    # Distribute remaining slots based on fractions
    deficit = num_phy - logcnt.sum(dim=1, keepdim=True)

    # Sort by marginal gain (approximation)
    scores = weight / (logcnt + 1).float()
    _, sorted_indices = torch.sort(scores, desc=True)

    # Vectorized allocation of remainder
    rank_idx = torch.arange(num_log).expand(num_layers,
      -1)
    mask = rank_idx < deficit
    logcnt.scatter_add_(1, sorted_indices, mask.long())

    return build_mapping(logcnt) # Helper omitted
```

Figure 5. Code evolution in the Load Balancing task. (a) The initial baseline uses inefficient iterative loops for both packing and replication. (b) The best code evolved by ContextEvolve introduces **Vectorized Snake Round-Robin** (Green Highlights) to maximize speed and **Proportional Apportionment** (Blue Highlights) to maximize load balance. These algorithmic breakthroughs resulted in a 33.3% improvement in the combined score.