Evolutionary Context Search for Skill Acquisition

Anonymous authors
Paper under double-blind review

Abstract

Large Language Models cannot reliably acquire new knowledge post-deployment—even when relevant text resources exist, models fail to transform them into actionable knowledge without retraining. Retrieval-Augmented Generation attempts to bridge this gap by surfacing relevant documents at inference time, yet similarity-based retrieval often fails to identify context that actually improves task performance. We introduce Evolutionary Context Search (ECS), an evolutionary method that searches context combinations using accuracy on a small development set, requiring only inference calls without weight updates. ECS moves beyond semantic similarity to discover non-obvious context pairings that significantly boost performance. Our empirical results show that ECS improves BackendBench by 27% and τ^2-bench airline by 5%. The evolved contexts are model-agnostic, as those evolved with Gemini-3-Flash transfer effectively to Claude-4.5-Sonnet and DeepSeek-V3.2. This suggests that ECS opens a path toward automated context discovery for skill acquisition—an efficient alternative to manual prompt engineering or costly fine-tuning.

1 Introduction

Updating the knowledge of Large Language Models (LLMs) to acquire new capabilities after the training cutoff remains a technical challenge (Onoe et al., 2023; Zhong et al., 2023; Yao et al., 2023; Li et al., 2023b). Domain-Specific Languages (DSLs) like CuTeDSL for GPU programming have comprehensive documentation, yet adapting LLMs to write code correctly in such high-resource but unseen languages cannot be done reliably (Kandpal et al., 2023; Gu et al., 2025). The core problem is not missing information, but effective methods to harness novel and diverse information sources to efficiently adapt the pretrained knowledge of an LLM to acquire the new target capability.

Existing approaches to skill acquisition incur substantial computational costs while struggling to obtain the required skill. Training-based methods, such as supervised finetuning (SFT) and reinforcement learning (RL) on curated data, are expensive due to their computational requirements, with additional engineering costs incurred by data collection and processing (Cottier et al., 2024). Moreover, given post-training requires weight access, such methods are naturally inapplicable to frontier, closed-source models. Current in-context approaches offer only partial solutions to training-based methods. Retrieval-Augmented Generation (RAG) (Lewis et al., 2020; Ram et al., 2023; Khandelwal et al., 2019) can equip base models with new knowledge at test-time without necessitating weight access, but similarity-based retrieval often fails because queries tend to be verbose, contain irrelevant context, or not task-specific (Li et al., 2023a; Petroni et al., 2020; Yoran et al., 2023). More generally, RAG is highly sensitive to arbitrary context ordering and requires considerable human engineering effort (Akkiraju et al., 2024), impairing the efficacy and viability of the technique.

In this work, we leverage the viewpoint that text-based prompt augmentations offers a flexible and effective framework for updating an LLM's knowledge, and develop a search method for accumulating the required context that avoids the issues of retrieval entirely. Our core insight is that optimal prompt construction for novel skill acquisition ought to be an evolutionary search process. Crucially, rather than relying on LLMs as evolutionary operators, we employ a simple genetic algorithm-style (Katoch et al., 2021) approach to evolve context combinations. We find this process to be highly efficient, typically requiring as few as 5 iterations to converge on high-utility results. This search process intuitively mirrors how humans learn new skills – collection of relevant documentation and other resources, assessing the fitness of a given resource by how it

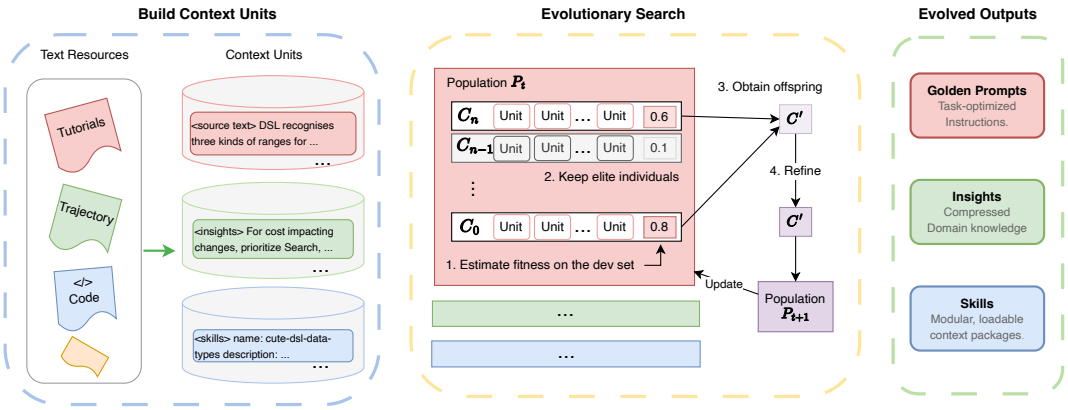

Figure 1: **Evolutionary Context Search.** Our method takes a population of text resources and evolves optimized contexts that confer the knowledge required to perform tasks in unseen domains. Each successive generation accumulates task-dependent, knowledge-rich information, effectively searching the corpora to obtain token-efficient contexts that enable novel skill acquisition in LLMs.

improves our understanding, using such resources to discover followup works, and so on recursively until we accumulate the required contextual information to attain the new capability.

Building on these insights, we introduce Evolutionary Context Search (ECS) (Figure 1), a framework that extracts knowledge from provided corpora through fitness-guided evolution rather than similarity-based retrieval. ECS takes a context-centric view: ECS converts text resources into context units and combines them into candidate contexts. These contexts are then evolved to improve performance on specific tasks. The evolved context can take multiple forms: raw documentation, condensed summaries/insights, or structured agent skills (Anthropic, 2025b). This approach transforms passive text resources into active teaching materials that progressively improve model performance. Our approach provides three connected benefits: 1) substantive performance gains beyond baseline methods as ECS evolves optimal contexts to provide base LLMs with the required new knowledge, 2) improved robustness to non-task-specific, verbose, or misleading queries, which markedly deteriorate the performance of retrieval-based methods, and 3) efficiency gains, as our method requires no weight or model access, or any training.

Our contributions are three-fold:

1. We propose Evolutionary Context Search (ECS), a framework that treats context seleciton as an optimization problem to maximize skill and knowledge acquisition from external resources.

2. We empirically demonstrate that ECS provides substantial improvements over baselines across diverse benchmarks, including τ^2-bench and BackendBench.

3. We show that the contexts discovered by ECS are model-agnostic and highly transferable, suggesting that ECS unlocks a new paradigm for injecting corpus knowledge into deployed LLMs.

2 Related Work

Knowledge updating. Existing approaches to update an LLM's knowledge to unseen information can be broken into two sets, gradient-based and in-context. Gradient-based approaches include supervised fine tuning and reinforcement learning from curated data, which incur persistent challenges due to catastrophic forgetting (Luo et al., 2025; Shi et al., 2025), not to mention substantial engineering and computational overhead caused by training and data curation. Parameter efficient fine tuning methods (Hu et al., 2022; Dettmers et al., 2023; Tian et al., 2024) typically trade computational overhead for performance (Chen et al., 2022; Biderman et al., 2024), while still requiring expensive data collection. Additionally, gradient based methods require model

weight access, thereby precluding application to powerful closed-source models. In-context approaches, on the other hand, resolve the need for weight access. Retrieval augmented generation (Lewis et al., 2020; Ram et al., 2023; Khandelwal et al., 2019) aims to equip the model with new knowledge through similarity based indexing, but is highly sensitive to arbitrary factors such as ordering (Liu et al., 2024a). Our approach builds off of in-context instruction provision, but automatically discovers populations of resources to optimally confer the required knowledge update without any manual, iterative prompt refinement.

Prompting. Optimal performance in specialized tasks is highly dependent on the prompting strategy (Zhou et al., 2022; Wang et al., 2022; 2023; Madaan et al., 2023). Recent studies avoid manual prompt engineering by learning prompts through optimizing continuous embeddings Li et al. (2024); Sinhababu et al. (2025); Liu et al. (2024b); Qin & Eisner (2021); Lester et al. (2021). Evolutionary methods have emerged as powerful alternatives for prompt optimization which avoid the need for gradients. Agrawal et al. (2025) learn prompts through sampling system-level trajectories and incorporating natural language reflection, while Guo et al. (2024) prompt LLMs to execute a fixed mutation and crossover process to produce offspring. Fernando et al. (2023) use an LLM to iteratively and self-referentially mutate prompts while improving the mutation prompts themselves. Our method differs from existing evolutionary methods in terms of both method and scope. Regarding method, we depart from LLMs as mutation and crossover operators (Guo et al., 2024; Fernando et al., 2023), which are ineffective when the model lacks the knowledge required to meaningfully manipulate task-related information. Instead, we perform mutation with probabilistic draws from the resource pool and crossover with shuffled concatenation, which broadens the exploration space. Regarding scope, we demonstrate evolution's capability as a search method to accumulate knowledge from diverse sources in unseen domains. Where prior work sought to utilize LLMs to iteratively rephrase task queries – adding instructions to show working, answer in full sentences etc – we consider the problem setting of evolving contexts from provided corpora for conferring the knowledge necessary to perform *new skills*. Hence our approach offers a new avenue towards evolutionary search for novel skill acquisition, moving beyond the evolution of LLM generated outputs towards evolution of knowledge bases themselves.

3 Evolutionary context search

We present ECS, a framework that evolves raw text resources into high-utility context for LLM skill acquisition. This section formalizes the context-evolution problem and details our pipeline, from the initial construction of context units to the specific GA-style operators used to optimize them for task performance.

3.1 Problem Formulation

We formalize context optimization as a search problem. Given text resources $\mathcal{D}$, a target language model $\mathcal{M}$ and a development task set $\mathcal{T}$, we seek an optimal context C^* that maximizes task performance of $\mathcal{M}$.

$$f(C; M, \mathcal{T}) = \mathbf{E}_{(x,y)\in\mathcal{T}}[\mathcal{L}\big(\mathcal{M}(x, C), y\big)] \tag{1}$$

where $\mathcal{L}$ is a task-specific scoring metric. We search for $C^* = \arg\max_{C\subseteq\mathcal{U}} f(C; \mathcal{M}, \mathcal{T})$, where $\mathcal{U} = g(\mathcal{D})$ is a set of context units derived from $\mathcal{D}$, each of which represents an atomic piece of knowledge that can be independently selected, combined, and refined. A candidate context C is then a structured combination of these units.

3.2 Algorithm Overview

ECS adapts to diverse tasks by constructing context units in multiple forms (i.e., $g(\mathcal{D})$): raw text from documentation, insights derived from long trajectories, or reusable agentic skills. Prior evolution-based approaches mutate model-generated content, for example rephrasing self-generated instructions, which inherently limits the search space to knowledge the model can already produce. By contrast, ECS draws mutations from provided external text resources, enabling the model to acquire genuinely missing information from its parameters.

Algorithm 1: Evolutionary Context Search

Require: Text resources $\mathcal{D}$, dev set $\mathcal{T}$, target model $\mathcal{M}$, population size N, generations T
Ensure: Optimal context C^*

$\mathcal{U} \leftarrow g(\mathcal{D})$ {raw text, insights, skills}
$P_0 \leftarrow \textsc{Initialize}(\mathcal{U}, N)$
for $t = 0$ **to** $T - 1$ **do**
  for all $C \in P_t$ **do**
    $s(C) \leftarrow \textsc{Evaluate}(C, \mathcal{M}, \mathcal{T})$
  end for
  $P_{\text{elite}} \leftarrow \textsc{SelectElite}(P_t, s)$
  $P_{t+1} \leftarrow \emptyset$
  for $j = 1$ **to** N **do**
    $C_a, C_b \leftarrow \textsc{Sample}(P_{\text{elite}}, s)$
    $C' \leftarrow \textsc{Crossover}(C_a, C_b)$
    $C' \leftarrow \textsc{Mutate}(C', \mathcal{U})$
    $C' \leftarrow \textsc{Refine}(C', \mathcal{T})$ {LLM-guided}
    $P_{t+1} \leftarrow P_{t+1} \cup \{C'\}$
  end for
end for
return $C^* \leftarrow \arg\max_{C \in P_T} \textsc{Evaluate}(C, \mathcal{M}, \mathcal{T})$

Algorithm 1 presents the complete ECS procedure. The algorithm operates in two phases: initialization and evolution. During initialization, we construct the initial pool $\mathcal{U}$ from text resources $\mathcal{D}$, extracting units at varying abstraction levels depending on the task setting — from verbatim source texts to distilled insights and reusable skills (Sec 3.3). We then sample an initial population P_0 of N candidate contexts by drawing units from $\mathcal{U}$ without replacement, ensuring broad coverage of the knowledge pool while maintaining comparable context lengths across candidates.

The evolution loop iteratively improves the population P_t. Each generation evaluates all candidates on the development set $\mathcal{T}$ and selects top performers as elite contexts P_{elite}. We sample parents fitness-proportionally from P_{elite} and produce offspring through crossover. Mutation then introduces variation by adding or replacing units to explore the full unit space $\mathcal{U}$. Mutation and crossover thereby jointly expand the population of performant contexts, as determined by those contexts contribution to advancing $\mathcal{M}$ performance on the task. Finally, LLM-guided refinement resolves logical contradictions inside each offspring. After T generations, we return the highest-performing context C^*.

3.3 Context Unit Construction

Different tasks require knowledge at different granularities: code generation benefits from exact syntax and precise documentation, while role-play benefits from abstracted principles. We therefore design context units at varying abstraction levels, allowing ECS to evolve units that span these differing levels of abstraction as necessitated by different tasks. We define three representative types below.

Source texts. These units preserve the precise syntax from source materials, which is vital for replicating exact phrasing or code patterns. For example, in Domain-Specific Language (DSL) tasks, we include complete code files from NVIDIA's CuTe tutorials, maintaining exact API usage patterns that the model must reproduce.

Insights. These units distill abstract principles from the provided text source. Typically, these are actionable rules that capture patterns the model should follow or pitfalls to avoid. For example, given failed trajectories from τ^2-bench (Barres et al., 2025), we prompt another model to analyze the errors and extracts rules such as "when processing refunds, retrieve the payment directly from the history rather than prompting the user."

Skills. These units encode reusable procedural knowledge as modular, callable actions. Each skill packages domain-specific instructions and multi-step workflows that the model can invoke on demand. We adapt the Agent Skills format (Anthropic, 2025b), which provides a structured representation for packaging procedural

knowledge. An example is "write-mha-cutedsl-kernel," which encapsulates the procedure for writing multi-head attention kernels in CuTe DSL. While skills offer a natural structure for procedural knowledge, naively including all available skills can degrade performance due to context distraction. ECS partially mitigates this by automatically curating task-relevant skills (see Section 4), though improving skill representation and invocation remains an open direction.

3.4 Evolutionary Operators

We now detail each evolutionary operator which jointly comprise our context search method.

Initialization. We initialize population P_0 of N candidates (typically 32) by sampling units from $\mathcal{U}$ uniformly without replacement. Each candidate context starts with a predetermined number of units, which we set based on task characteristics: tasks with long units (e.g., complete code files) use fewer units per context, while task with short units (e.g., concise insights) use more. This maintains comparable context lengths across candidates while preserving diversity among individuals in the initial population.

Fitness Evaluation. We score each context C on the development set $\mathcal{T}$ by querying the LLM $\mathcal{M}$ with C as context across all tasks in the development set $\mathcal{T}$. The fitness $s(C)$ equals the task success rate, normalized to $[0, 1]$. We use a single rollout per task, which we find sufficient in practice, though additional rollouts could further reduce variance from stochastic $\mathcal{M}$ outputs.

Selection. Selection pressure drives the population toward higher-performing contexts. We first select the top fraction (e.g., 60%) of contexts as the elite set P_{elite}. We then repeatedly sample parent pairs from this elite set using fitness-proportional selection until we generate N offspring for the next generation. For any context C, its selection probability p_C equals its fitness normalized by the elite set's total fitness, $p_C = \frac{s(C)}{\sum_{K \in P_{elite}} s(K)}$. This focuses reproduction on top performers while maintaining variation within elites.

Crossover. Crossover combines units from two parent contexts C_a and C_b to produce one offspring. We concatenate all units from both parents; if the combined set exceeds the maximum context size, we randomly sample from the concatenated context unit pool. This allows offspring to inherit complementary information from both parents while respecting context length constraints.

Mutation. Mutation introduces variation controlled by a per-context mutation rate (default is 0.1). When mutation triggers, we sample a new unit from the full pool $\mathcal{U}$. If the context has not reached its maximum size, we add the new unit; otherwise, we replace a randomly selected existing unit. This operator ensures the algorithm explores the entire unit space and can escape local optima.

3.5 LLM-Guided Refinement

After mutation, an LLM reviews each offspring context to identify and resolve logical contradictions. The LLM receives the offspring context and instructions to detect inconsistencies; in principle, failed task examples could also be provided to guide refinement, though we omit this for simplicity in our experiments. This refinement step addresses a limitation of blind recombination: merged units may contain conflicting guidance. The LLM identifies contradictions and either removes or resolves the conflicts.

3.6 Convergence Analysis

To explain the rapid convergence observed in our experiments, we analyze an idealized elitist variant of ECS that captures its core search dynamics. The analysis abstracts away population-level crossover and LLM-guided refinement, and focuses on the core mechanism shared with our method: fitness-guided selection over fixed-size context sets with random replacement mutation. Let $\mathcal{U}$ be a pool of N context units, let $T^\star \subseteq \mathcal{U}$ denote an optimal target set of size r, and let each candidate context $S_t \subseteq \mathcal{U}$ have capacity k, where $r \leq k < N$. Define $X_t = |S_t \cap T^\star|$ as the overlap between the current context and the target set. Under a monotone utility assumption in which adding a target unit strictly improves expected task performance

beyond bounded evaluation noise, elitist selection makes $\{X_t\}$ non-decreasing. Thus, the search process is a pure birth Markov process.

Proposition 3.1 (Log-linear convergence)**.** *Assume fixed-size random replacement mutation and elitist selection, with excess capacity $k \geq (1+\gamma)r$ for some constant $\gamma > 0$. If the utility has the form $U(S) = g(|S \cap T^\star|) + b(S)$, where $|b(S)| \leq \epsilon$ and $g(i+1) - g(i) \geq \beta > 2\epsilon$ for all $i \in \{0, \ldots, r-1\}$, then the expected time to recover all r target units satisfies*

$$\mathbb{E}[\tau_r] = \mathcal{O}(N \log r).$$

The result connects ECS to classical analyses of elitist evolutionary algorithms and absorbing Markov chains (Rudolph, 1994; Droste et al., 2002), which establish logarithmic-factor convergence rates for monotone pseudo-Boolean optimization. In our setting, the proposition suggests that when useful context units provide reliably positive marginal utility, fitness-guided context search can identify high-utility combinations in few generations. We provide the full proof and discussion in Appendix D.

4 Experiments

We empirically validate the benefits of ECS in skill acquisition. We evaluate our method on coding in unseen domain-specific languages with BackendBench (Saroufim et al., 2025) and on multi-turn agentic user assistance with τ^2-Bench (Barres et al., 2025), using Gemini-3-Flash (Gemini Team, 2025a) as the base model for context search. We also demonstrate cross-model transferability by applying the discovered contexts to Claude-4.5-Sonnet (Anthropic, 2025a) and DeepSeek-V3.2 (Liu et al., 2025). In addition, Appendix F demonstrates that SFT remains ineffective for knowledge injection in data-scarce regimes.

4.1 Experimental Setup

Baselines. We evaluate ECS against several baselines. We implement RAG baselines with LlamaIndex (Liu, 2022), varying two primary axes: chunking strategy and retrieval method. Chunking strategies include fixed-size splitting (**Chunk**; 1,024 tokens with 200-token overlap) and **AST**-based parsing, which segments code at semantic boundaries. Retrieval methods include **Dense** (similarity over OpenAI text-embedding-3-small embeddings (OpenAI, 2024)), **BM25** (sparse keyword matching), and **Hybrid** (reciprocal rank fusion of Dense and BM25). We sweep $k \in \{5, 10, 20\}$ for Chunk + Dense and find $k = 10$ performs best, which we use for all RAG configurations. Beyond RAG, we include **Full Context**, which loads all available documentation into the context window, and **Random Sample**, which randomly selects the same number of context units as ECS.

Tasks. We evaluate our method on two challenging benchmarks: kernel coding in new Domain-Specific Language (DSL) and multi-turn agentic user assistance.

BackendBench (CuTeDSL) (Saroufim et al., 2025) focuses on testing whether models can generate correct GPU kernels in various DSLs (Triton, CUDA, CuTeDSL). Among these, we choose CuTeDSL to showcase ECS capacity to convert newly released textual tutorials into knowledge-rich context capable of guiding the model. CuTeDSL is NVIDIA's Python DSL built on CUDA Templates for Linear Algebra Subroutines (CUTLASS) (NVIDIA, 2026), where text resources are provided by tutorial examples in the repository. We randomly select 20 core PyTorch operators, each with around 8–12 test cases from PyTorch's operator information (OpInfo) suite. Appendix B.1 details these operators. We run 3 evaluations and report the average correctness rate.

τ^2-Bench (Airline Domain) (Barres et al., 2025) evaluates conversational agents on completing user requests through multi-turn interaction, while adhering to domain-specific policies which may conflict with user requests. The airline domain presents customer service tasks – booking modifications, cancellations, and policy inquiries – that must be completed while maintaining airline policy standards. We obtain text resources by collecting trajectories from GPT-5.2 (OpenAI, 2025) and Gemini-3-Pro (Gemini Team, 2025b) on the official training set, then prompting Gemini-3-Flash to extract insights from these trajectories. Following Barres et al. (2025), we use GPT-4.1-2025-04-14 (Achiam et al., 2023) as the user simulator. Pass^k for

$k \in \{1, 2, 3\}$ measures the rate at which all k trials succeed. We evaluate each configuration 3 times with 3 trials, yielding 9 runs total.

ECS Configuration. We run ECS for 5 generations (10 for τ^2-Bench) with a population size of 32, selecting the top 60% as elites with a mutation rate of 0.1. Each context is limited to a maximum of 10 units, drawn from a pool of 85 source documents for BackendBench and 60 extracted insights for τ^2-Bench. Fitness is evaluated on 10 development samples for BackendBench and on the 30 official training tasks for τ^2-Bench. In our experiments, $\mathcal{M}$ is Gemini-3-Flash, and we use Gemini-3-Flash itself for refinement (the Refine step in Algorithm 1), keeping the entire pipeline within a single model so that any reported gains are not driven by an auxiliary, more capable refiner. Train/dev/test splits are kept strictly disjoint for both benchmarks; full split documentation is in Appendix A, significance tests over n=3 seeds in Appendix B.3, and hyperparameter sensitivity in Appendix C.1.

4.2 Main Results

Observation 1: Evolutionary Context Search constructs more effective context than retrieval-based approaches. ECS consistently outperforms standard retrieval baselines across both the DSL kernel coding and agentic tasks.

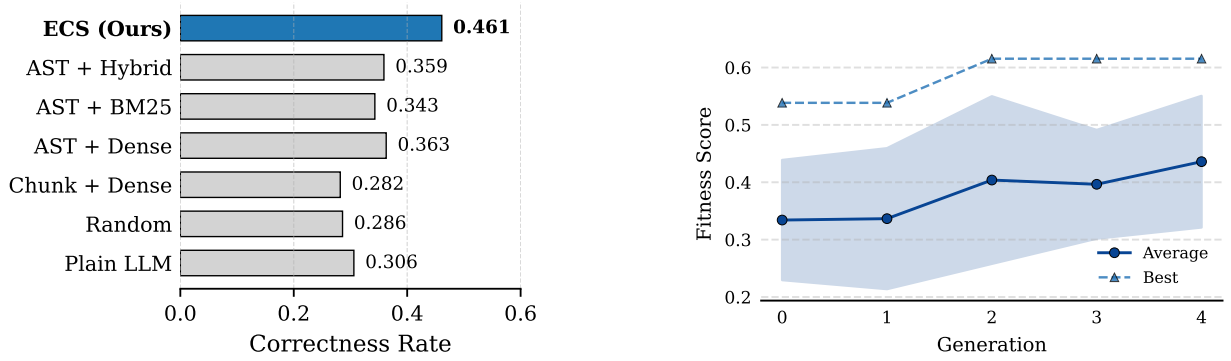

Figure 2: **Left:** Performance comparison on BackendBench. ECS achieves a 27% relative improvement over AST+Dense. Detailed results in Appendix B.2. **Right:** Fitness score during the evolutionary search process.

Regarding BackendBench, where the model must master CuTeDSL using tutorial-based coding resources, Figure 2 (left) shows that ECS achieves a correctness rate of 0.461, outperforming the strongest retrieval baseline (AST + Dense) by 27% relative improvement. We also illustrate the training dynamics in Figure 2 (right). The rapid improvement in the fitness curve is consistent with the log-linear convergence behavior predicted by Proposition 3.1 under monotone marginal utility assumptions. Beyond this, several patterns emerge. First, the chunking strategy is critical: all AST-based methods consistently outperform Chunk + Dense, confirming that preserving semantic boundaries matters for code documentation. Second, retrieval method choice has limited impact—Dense, Hybrid, and BM25 perform similarly when paired with AST. Third, naive approaches can hurt: both Chunk + Dense and Random underperform the Plain LLM baseline. Since Random uses the same context budget as ECS, this confirms that arbitrary selection introduces noise, which degrades performance. This concurs with existing findings that irrelevant context deteriorates RAG's performance relative to the base model (Yoran et al., 2023), implying that RAG is dependent on careful curation. These results expose a core limitation of retrieval: optimizing chunk-level relevance rather than compositional coherence. ECS addresses this by evolving context combinations that maximize task performance.

This superior performance is further validated on the agentic τ^2-*Bench* in Table 1, where ECS achieves 0.756 on Pass^1, outperforming the strongest baseline (Full Context at 0.717). Two observations stand out. First, the gap widens under stricter metrics: on Pass^3, ECS maintains 0.667 while Plain LLM and BM25 degrade sharply. This indicates that curated contexts enable more consistent policy adherence across repeated

trials. Second, Full Context outperforms retrieval-based methods, suggesting that broader coverage matters for policy-heavy tasks—yet, ECS surpasses Full Context while using far less context, demonstrating that selective curation beats brute-force inclusion. This reveals a further benefit of ECS, which is that our method constructs efficient contexts that confer the required knowledge without context bloat.

Table 1: **Comparison on τ^2-Bench.** ECS significantly outperforms baselines across all Pass metrics.

Method	**Pass^1**	**Pass^2**	**Pass^3**
Plain LLM	0.622	0.489	0.400
Random	0.650	0.556	0.500
BM25	0.641	0.550	0.483
Full	0.717	0.628	0.550
ECS (Ours)	**0.756**	**0.700**	**0.667**

Observation 2: The evolved context is highly transferable across different models, as it encodes semantically meaningful information. To assess the generalizability of ECS beyond the base model, we evaluate contexts evolved with Gemini-3-Flash on two powerful held-out models: Claude-4.5-Sonnet and DeepSeek-V3.2.

As shown in Figure 3, for *BackendBench* ECS exhibits strong generalization. With Claude-4.5-Sonnet, the evolved contexts achieve 0.611 correctness rate, surpassing the strong AST+Dense baseline at 0.564. Notably, with DeepSeek-V3.2, where standard retrieval fails to provide meaningful gains (0.065 vs 0.031 plain), ECS unlocks significant capapbilities, reaching 0.223, a 7x improvement over the plain baseline. We provide further analysis on this context transfer in Section 5.2.

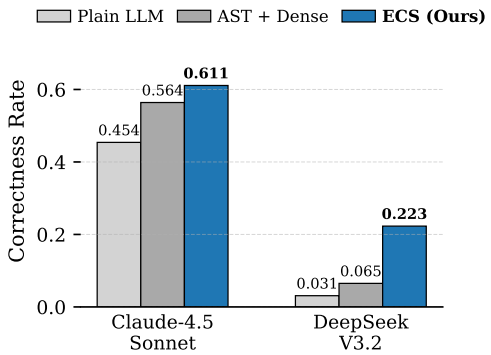

Figure 3: **Transferability to unseen models (BackendBench).** Contexts evolved by ECS transfer effectively to models not used during evolution. On DeepSeek-V3.2, where standard retrieval (AST+Dense) fails to provide meaningful gains, ECS unlocks significant capability, yielding a significant improvement.

This strong generalization performance holds for τ^2-*Bench* (Table 2). With Claude-4.5-Sonnet, ECS matches Full on Pass^1 and outperforms it on stricter metrics (Pass^3: 0.583 vs. 0.500), indicating that ECS effectively filters noise that otherwise distracts capable models. With DeepSeek-V3.2, ECS reaches Pass^1 comparable to Full (0.600 vs. 0.633) using 6× fewer tokens (848 vs. 5491), and matches Plain on Pass^2/Pass^3. These results demonstrate that ECS captures transferable contexts enabling instant skill acquisition—even when evolved with a smaller model (Gemini-3-Flash) and then transferred to larger, more capable models. Moreover, this result suggests the possibility of using ECS as an data curation process for high-utility SFT data.

Observation 3: Evolutionary Context Search enables effective skill-based augmentation by automatically curating task-relevant skills. Agentic skills represent an emerging paradigm for extending LLM capabilities (e.g., Claude Skills), but best practices for skill selection remain an open challenge. To evaluate ECS in this setting, we extract 75 skills from the CuTeDSL documentation, each corresponding to an example code snippet. From our experiments, we find that naively including all available skills degrades performance (0.283) compared to the plain LLM baseline (0.306), due to context distraction caused by irrelevant snippets. By contrast, ECS successfully filters this noise, achieving the highest correctness rate (0.310). While this falls short of evolving directly on source text (0.461), it demonstrates that ECS can serve as a complementary technique for skill-based systems—transforming noisy skill pools into effective context through evolutionary curation.

Table 2: **Transferability on τ^2-Bench.** With Claude-4.5-Sonnet, ECS matches the Full baseline on Pass^1 and significantly outperforms it on stricter metrics (Pass^2, Pass^3). Notably, ECS achieves comparable performance to the Full baselines with substantially fewer tokens, reducing usage from 5491 to 848, a 6× reduction.

	Claude-4.5-Sonnet			**DeepSeek-V3.2**		
Metric	**Plain**	**Full**	**ECS**	**Plain**	**Full**	**ECS**
Pass^1	0.600	0.678	**0.678**	0.583	**0.633**	0.600
Pass^2	0.506	0.561	**0.617**	0.478	**0.500**	0.472
Pass^3	0.450	0.500	**0.583**	0.417	0.417	0.417
# Tokens	0	5491	848	0	5491	848

4.3 Comparison with Additional Baselines

We further compare ECS against three stronger paradigms that go beyond static retrieval: dynamic RAG, LLM-based prompt optimization, and direct LLM context curation. Together, these baselines isolate whether ECS's gains come from the evolutionary search itself rather than from retrieval engineering, prompt rewriting, or unsupervised LLM reasoning over the corpus.

Open-source evaluation with dynamic RAG (FLARE). FLARE (Jiang et al., 2023) is a representative dynamic RAG method that adaptively decides when and what to retrieve during generation. Because FLARE requires access to output-token log-probabilities, which is unavailable for Gemini-3-Flash, we evaluate it on BackendBench using the open-source Qwen3.5-122B-A10B (Yang et al., 2025) model as both the target and the refinement model for ECS. This setting also serves as a sanity check that ECS is not specific to closed-source frontier models.

Table 3: **Open-source comparison on BackendBench with Qwen3.5-122B-A10B.** FLARE improves over Plain but falls behind standard dense RAG due to verbose, multi-part queries. ECS continues to lead by a substantial margin.

Method	**Correctness Rate**
Plain LLM	0.063
FLARE	0.198
AST + Dense RAG	0.227
ECS (Ours)	**0.278**

FLARE (0.198) improves over the Plain baseline (0.063) but falls slightly behind standard dense RAG (0.227). On inspection, FLARE still suffers from the verbose-query issue: BackendBench's official evaluation wraps each operator request in a long system+user prompt, which dominates the retrieval signal regardless of when retrieval is triggered. ECS (0.278) preserves its lead by sidestepping query-conditional retrieval entirely.

LLM-based prompt optimization (OPRO). OPRO (Yang et al., 2024) uses an LLM to iteratively propose and select better prompts from prior trajectories. For a fair comparison on τ^2-Bench, we provide OPRO with the same insight pool as ECS for seeding and match the compute budget. As shown in Table 4, OPRO improves over Plain LLM but consistently trails ECS at every Pass level. The widening gap on stricter metrics (Pass^3: 0.667 vs. 0.500) suggests that rewording or reordering a fixed instruction set provides limited room when the underlying knowledge requirements span heterogeneous policies.

Iterative LLM context curation. A natural concern is whether ECS's gains come from the evolutionary framework itself or simply from giving an LLM extended access to the corpus. To test this, we let Gemini-3-Flash directly perform iterative context curation: it sees the full insight pool and the training split, and is prompted to iteratively retain useful units up to a 320-iteration budget (matched to ECS). As Table 4 shows,

this baseline improves over Plain LLM (Pass^1: 0.678 vs. 0.622) but still falls below ECS on every Pass level. Concentrating the search in a single LLM trajectory loses the population-level diversity that ECS maintains across candidates.

Table 4: **Stronger baselines on τ^2-Bench.** Both OPRO and direct LLM context curation improve over Plain but fall short of ECS, confirming that the gains stem from population-level evolutionary search rather than from prompt rewriting or single-trajectory LLM reasoning over the corpus.

Method	**Pass^1**	**Pass^2**	**Pass^3**
Plain LLM	0.622	0.489	0.400
OPRO	0.694	0.578	0.500
LLM Curation (Flash)	0.678	0.578	0.533
ECS (Ours)	**0.756**	**0.700**	**0.667**

5 Analysis and Ablation

In this section, we examine the properties of ECS. We first analyze the evolved context qualitatively, then study how models utilize contexts. Finally, we conduct ablation studies and analyze computational costs.

5.1 Analysis of evolved context.

Figure 4 presents the context discovered with Gemini-3-Flash from the CuTeDSL code tutorial. From 85 available documents, ECS selected a combination of 7, yet these form a coherent stack spanning multiple abstraction levels.

(a) hopper/fmha.py `[Kernel]` **2,540 lines (58%)**

Role: Fused MHA with TMA + TensorCore

```
# Q*K^T, softmax, softmax(Q*K^T)*V fused
for j in cutlass.range_constexpr(
    cute.size(acc_qk_mn, mode=[1])):
    acc_qk_mn[i, j] = cute.math.exp2(
        scale_softmax_log2 * acc_qk_mn[i, j] - scale_max,
        fastmath=True)
```

(b) tensorop_gemm.py `[Kernel]` **1,012 lines (23%)**

Role: Dense GEMM for Ampere architecture

```
# Creates MMA atom with 16x8x16 shape for MNK
op = cute.nvgpu.warp.MmaF16BF16Op(
    self.ab_dtype, self.acc_dtype, self.mma_inst_shape)
tC = cute.make_layout(self.atom_layout_mnk)
tiled_mma = cute.make_tiled_mma(op, tC, permutation_mnk)
```

(c) jit_argument.py `[FFI]` **320 lines (7%)**

Role: C-struct tensor interface via LLVM

```
# FFI: Extract pointer from C-struct
ptr_val = llvm.extractvalue(
    llvm.PointerType.get(), self, [0], loc=loc, ip=ip)
return cute.make_ptr(cutlass.Float32, ptr_val)
```

Figure 4: **Core components of the BackendBench evolved context.** The figure illustrates the three most significant code snippets: **(a)** A Fused MHA kernel targeting the NVIDIA Hopper architecture (58% of context); **(b)** a dense GEMM kernel configuration for Ampere Tensor Cores (23%); and **(c)** a low-level FFI interface managing LLVM pointer extraction (7%).

At the kernel layer, `hopper/fmha.py` provides a comprehensive example of a fused mega-kernel—contributing TMA-based memory transfers, warp-specialized execution, and tensor core MMA(Matrix Multiply-Accumulate) patterns, while also demonstrating the `cute.math.*` API for arithmetic operations. The second kernel example, `tensorop_gemm.py`, shows how to construct tiled MMA operations, covering atom layout configuration and threadblock rasterization. Finally, at the FFI (Foreign Function Interface layer), `jit_argument.py` defines

C-struct tensor interfaces, enabling data passing between Python and compiled kernels. This layered composition suggests that ECS does not simply identify isolated code snippets, but in fact accumulates coherent architectural patterns that span from kernel implementation to interfaces.

5.2 Context utilization across models.

(a) Evolved Context
hopper/fmha.py (lines 1372–1384)

```
for j in range_constexpr(...):
  acc_qk_mn[i,j] = cute.math.exp2(
    scale_softmax_log2
    * acc_qk_mn[i,j]
```

`cute.math.*` pattern

(b) RAG Baseline
DeepSeek-V3.2 generates wrong API

```
input_val = gA[tidx]
result = math.atan(input_val)
```

× Fails: stdlib on symbolic value

(c) With Evolved Context
DeepSeek-V3.2 generalizes correctly

```
input_val = gA[tidx]
# CORRECT: DSL intrinsic
result = cute.math.atan(input_val)
```

✓ Works: DSL intrinsic

Operator	RAG Baseline	Evolved Context	Knowledge Transferred
`atan.default`	0% (0/1)	**100%** (1/1)	`cute.math.atan()`
`atan2.default`	0% (0/9)	**88.9%** (8/9)	`cute.math.atan2()`
`div.Tensor`	0% (0/18)	**88.9%** (16/18)	`cute.math.*` pattern
Overall (20 ops)	6.5%	**22.3%**	3.4× improvement

Figure 5: **Cross-model context transfer from Gemini-3-Flash to DeepSeek-V3.2 on BackendBench. Top:** (a) Evolved context demonstrates the `cute.math.*` pattern. (b) Without this context, DeepSeek-V3.2 incorrectly uses Python's `math.atan()`, causing JIT compilation failures. (c) With evolved context, DeepSeek-V3.2 correctly generalizes to DSL intrinsics. **Bottom:** Quantitative results show evolved context improves pass rate from 6.5% to 22.3% (3.4×); e.g., `div.Tensor` passes 16 out of 18 test cases.

We analyze how exactly the evolved context enables knowledge transfer across models. Figure 5 illustrates one specific example for BackendBench. The evolved context contains `fmha.py`, which has the `cute.math.exp2()` function for softmax operation. Although this example never mentions `atan` or other trigonometric functions, it implicitly teaches the correct API pattern: math operations in CuTeDSL require DSL intrinsics under `cute.math.*`, not Python's standard library. When DeepSeek-V3.2 receives this context, it extracts this structural pattern and generalizes it to unseen operators, correctly generating `cute.math.atan()` despite never observing this specific function in the provided examples.

In contrast, RAG retrieval for the `atan` operator returns semantically relevant but functionally useless context: GEMM kernels, elementwise addition, and tensor utilities—none of which contain any `cute.math.*` calls. Without working examples, DeepSeek falls back to Python's `math.atan()`. This fails because inside a `@cute.kernel`, tensor elements are symbolic `ArithValue` objects, not concrete Python values. Only DSL intrinsics, such as `cute.math.atan()`, are recognized by the JIT compiler and translated to CUDA device code. This demonstrates that ECS selects context containing transferable structural patterns, whereas RAG retrieves topically similar code that lacks the critical details needed for correct generation.

5.3 Ablation Study

To understand the contribution of each component, we evaluate ECS variants with fitness-guided selection, mutation, and refinement individually removed. Figure 6 shows the results. Fitness-guided selection is critical: removing it causes the largest degradation on both benchmarks, with BackendBench dropping to 0.286 (below Plain LLM). Mutation also proves essential, reducing performance on both tasks when removed. Interestingly, refinement exhibits domain-dependent behavior: on BackendBench, its removal has negligible impact (0.461 vs 0.458), whereas on τ^2-Bench it causes substantial degradation (Pass^3: 0.667 vs 0.488). This aligns with our earlier analysis, LLMs struggle to filter tutorial coding samples and tend to retain all of them, while insights often contain explicit contradictions (e.g., conflicting refund rules) that LLM can readily identify.

5.4 Computational cost.

Search Cost. ECS incurs an initial computational cost to discover the optimal context C^*. In our experiments, the search budget was limited to $T = 10$ generations with a population size of $N = 32$, resulting

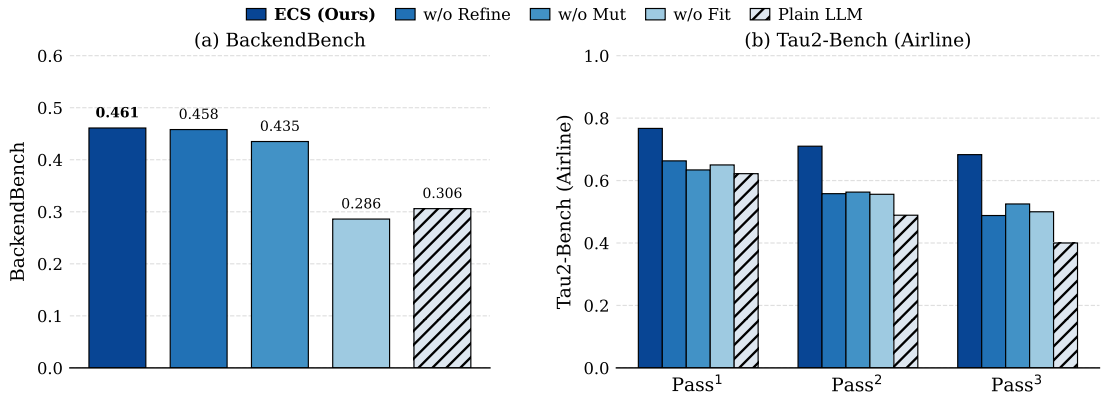

Figure 6: **Ablation study.** We evaluate variants by removing components. *Fitness* and *Mutation* are universally critical. *Refinement* is domain-dependent: essential for resolving conflicting agentic policies (τ^2-Bench) but with negligible impact on BackendBench.

in approximately 320 evaluations on the development set (fewer on BackendBench, which uses $T = 5$). While this exceeds the setup cost of standard RAG, which requires computing embeddings for dense retrieval, it is orders of magnitude cheaper than typical SFT or RL workflows, which often necessitate tens of thousands of rollouts to stabilize policy updates Shao et al. (2024). Crucially, ECS requires only black-box inference calls, avoiding the memory overhead of gradient updates and optimizer states required by gradient-based methods.

Inference Efficiency. Once the optimal context is discovered, ECS offers significant efficiency gains over retrieval baselines during deployment. Because the evolved context C^* remains static across all incoming queries for a given task, it is fully compatible with Context Caching (KV-Caching). In contrast, RAG systems retrieve different context chunks for every unique query, preventing the model from hitting the cached prompt prefix. Current industry pricing for cached context (e.g. Claude 4.5) charges approximately 10% of the standard input token price for cache hits Anthropic (2024). Furthermore, caching eliminates the prefill calculation for the context, significantly reducing Time-To-First-Token. Thus, while ECS requires an upfront search investment, it largely reduces the recurring deployment cost for the context portion compared to dynamic retrieval methods.

6 Conclusions

Summary. We introduce Evolutionary Context Search (ECS), a method that reframes knowledge acquisition as an evolutionary search over text resources. ECS achieves a 27% relative improvement on BackendBench over the strongest RAG baseline, and on τ^2-Bench surpasses the non-evolutionary baseline (Full Context) by 5%. Furthermore, contexts evolved using Gemini-3-Flash transfer effectively to Claude-4.5-Sonnet and DeepSeek-V3.2, notably yielding a 7× improvement on DeepSeek where RAG provides negligible gains.

Limitation. Several factors bound ECS's applicability. First, the development set must be representative of the deployment distribution, since task accuracy on it drives fitness; non-representative dev sets risk overfitting the evolved context. Second, the transfer guarantee rests on the aligned-marginal-utility condition (Appendix E), which can weaken when source and target models benefit from disjoint structural patterns. Third, the upfront search cost amortizes only when the evolved context is reused across many queries; in single-query regimes, lightweight retrieval may still be preferable. Finally, "needle-in-the-haystack" corpora—where useful units are extremely rare relative to the pool—can slow random-mutation convergence; LLM-guided population initialization is a promising refinement.

Future Work. Most notably, the transferability of our results suggests that ECS can serve as an automated data curation mechanism that helps SFT. An iterative pipeline, where ECS discovers optimal demonstrations and SFT internalizes them, forms a robust paradigm for enabling open-source models to master new skills with minimal human intervention.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Rama Akkiraju, Anbang Xu, Deepak Bora, Tan Yu, Lu An, Vishal Seth, Aaditya Shukla, Pritam Gundecha, Hridhay Mehta, Ashwin Jha, et al. Facts about building retrieval augmented generation-based chatbots. *arXiv preprint arXiv:2407.07858*, 2024.

Anthropic. Prompt caching: Reducing latency and cost. `https://platform.claude.com/docs/en/build-with-claude/prompt-caching`, 2024. Accessed: 2025-01-27. States that cache reads are charged at 0.1x of the base input token price.

Anthropic. Claude sonnet 4.5 system card. System card, Anthropic, September 2025a. URL `https://www-cdn.anthropic.com/963373e433e489a87a10c823c52a0a013e9172dd.pdf`.

Anthropic. Agent skills. `https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview`, 2025b. Accessed: 2025-01-27.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL `https://arxiv.org/abs/2506.07982`.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. Revisiting parameter-efficient tuning: Are we really there yet? *arXiv preprint arXiv:2202.07962*, 2022.

Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1–2):51–81, 2002.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Gemini Team. Gemini 3 Flash model card. Model card, Google DeepMind, December 2025a. URL `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf`. Published December 2025.

Gemini Team. Gemini 3 Pro model card. Model card, Google DeepMind, November 2025b. URL `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf`. Model card published November 2025.

Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. On the effectiveness of large language models in domain-specific code generation. *ACM Transactions on Software Engineering and Methodology*, 34(3):1–22, 2025.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large language models struggle to learn long-tail knowledge. In *International conference on machine learning*, pp. 15696–15707. PMLR, 2023.

Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, 80(5):8091–8126, 2021.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Chengzhengxu Li, Xiaoming Liu, Zhaohan Zhang, Yichen Wang, Chen Liu, Yu Lan, and Chao Shen. Concentrate attention: Towards domain-generalizable prompt optimization for language models. *Advances in Neural Information Processing Systems*, 37:3391–3420, 2024.

Daliang Li, Ankit Singh Rawat, Manzil Zaheer, Xin Wang, Michal Lukasik, Andreas Veit, Felix Yu, and Sanjiv Kumar. Large language models with controllable working memory. In *Findings of the association for computational linguistics: ACL 2023*, pp. 1774–1793, 2023a.

Zhoubo Li, Ningyu Zhang, Yunzhi Yao, Mengru Wang, Xi Chen, and Huajun Chen. Unveiling the pitfalls of knowledge editing for large language models. *arXiv preprint arXiv:2310.02129*, 2023b.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Jerry Liu. LlamaIndex, 11 2022. URL `https://github.com/jerryjliu/llama_index`.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *AI Open*, 5:208–215, 2024b.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback, 2023. *URL https://arxiv. org/abs/2303.17651*, 2023.

NVIDIA. *NVIDIA CUTLASS Documentation (CUTLASS 4.3.5)*. NVIDIA, January 2026. URL `https://docs.nvidia.com/cutlass/latest/`. Online documentation for CUTLASS 4.3.5 (Jan 2026); accessed 2026-01-28.

Yasumasa Onoe, Michael Zhang, Shankar Padmanabhan, Greg Durrett, and Eunsol Choi. Can lms learn new entities from descriptions? challenges in propagating injected knowledge. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5469–5485, 2023.

OpenAI. New embedding models and api updates. `https://openai.com/index/new-embedding-models-and-api-updates`, 2024. Announcing text-embedding-3-small and text-embedding-3-large.

OpenAI. Update to gpt-5 system card: Gpt-5.2. System card, OpenAI, December 2025. URL `https://cdn.openai.com/pdf/3a4153c8-c748-4b71-8e31-aecbde944f8d/oai_5_2_system-card.pdf`. Dated December 11, 2025.

Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. How context affects language models' factual predictions. *arXiv preprint arXiv:2005.04611*, 2020.

Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. *arXiv preprint arXiv:2104.06599*, 2021.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.

Mark Saroufim, Jiannan Wang, Bert Maher, Sahan Paliskara, Laura Wang, Shahin Sefati, and Manuel Candales. Backendbench: An evaluation suite for testing how well llms and humans can write pytorch backends, 2025. URL `https://github.com/meta-pytorch/BackendBench`.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 58(5):1–42, 2025.

Nilanjan Sinhababu, Pabitra Mitra, and Debasis Ganguly. Soft prompt improves direct search responses. In *Proceedings of the 17th annual meeting of the Forum for Information Retrieval Evaluation*, pp. 79–87, 2025.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Cheng-Zhong Xu. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37:9565–9584, 2024.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*, 2023.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. Making retrieval-augmented language models robust to irrelevant context. *arXiv preprint arXiv:2310.01558*, 2023.

Zexuan Zhong, Zhengxuan Wu, Christopher D Manning, Christopher Potts, and Danqi Chen. Mquake: Assessing knowledge editing in language models via multi-hop questions. *arXiv preprint arXiv:2305.14795*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Data Splits

We keep the data used to drive evolutionary search strictly disjoint from the data used to report final scores in both benchmarks.

τ^2-Bench (airline). We use the airline domain, which provides two official splits: a 30-task train split and a 20-task test split. The 30 train tasks are used solely for fitness evaluation during search and for constructing insight units (including the trajectory pool used by another model to distill rules). The 20 test tasks are used *exclusively* for the numbers reported in the main paper; no test task is ever shown to ECS during search.

BackendBench (CuTeDSL). The 20 PyTorch operators in our evaluation suite (Appendix B.1) are used exclusively for reporting. For fitness evaluation during ECS search we use 10 development samples drawn from a disjoint development pool of operators not included in the evaluation suite; no test case from the 20 reporting operators is ever shown to ECS during search.

B BackendBench Details

B.1 Evaluation Operators

We evaluate our method on 20 PyTorch operators from the BackendBench benchmark, spanning three categories: trigonometric functions (8), arithmetic operations (4), and linear algebra primitives (8). These operators represent a diverse set of computational patterns commonly encountered in deep learning workloads. Table 5 provides detailed descriptions of each operator.

B.2 Full BackendBench Results

Table 5: PyTorch operators used in BackendBench evaluation. We evaluate 20 operators across three categories: trigonometric functions (8 ops), arithmetic operations (4 ops), and linear algebra primitives (8 ops).

Category	**Operator**	**Signature**	**Description**
Trigonometric Functions			
	`acos`	$f : [-1, 1] \to [0, \pi]$	Computes element-wise inverse cosine (arccosine) of the input tensor.
	`acosh`	$f : [1, \infty) \to [0, \infty)$	Computes element-wise inverse hyperbolic cosine: $\ln(x + \sqrt{x^2 - 1})$.
	`asin`	$f : [-1, 1] \to [-\frac{\pi}{2}, \frac{\pi}{2}]$	Computes element-wise inverse sine (arcsine) of the input tensor.
	`asinh`	$f : \mathbb{R} \to \mathbb{R}$	Computes element-wise inverse hyperbolic sine: $\ln(x + \sqrt{x^2 + 1})$.
	`atan`	$f : \mathbb{R} \to (-\frac{\pi}{2}, \frac{\pi}{2})$	Computes element-wise inverse tangent (arctangent) of the input tensor.
	`atan2`	$f : \mathbb{R}^2 \to [-\pi, \pi]$	Computes element-wise two-argument arctangent of y/x, using signs to determine quadrant.
	`atanh`	$f : (-1, 1) \to \mathbb{R}$	Computes element-wise inverse hyperbolic tangent: $\frac{1}{2} \ln \frac{1+x}{1-x}$.
	`ceil`	$f : \mathbb{R} \to \mathbb{Z}$	Rounds each element to the smallest integer greater than or equal to the input.
Arithmetic Operations			
	`div`	$(a, b) \mapsto a/b$	Computes element-wise division. Supports both integer and floating-point semantics.
	`div_mode`	$(a, b, \text{mode}) \mapsto a/b$	Element-wise division with explicit rounding: `trunc` or `floor` mode.
	`fmod`	$(a, b) \mapsto a - b \cdot \text{trunc}(a/b)$	C-style remainder (truncated division). Result has same sign as dividend.
	`remainder`	$(a, b) \mapsto a - b \cdot \text{floor}(a/b)$	Python-style remainder (floored division). Result has same sign as divisor.
Linear Algebra Primitives			
	`addmm`	$\beta M + \alpha(A@B)$	Matrix-matrix multiply with accumulation. $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$, $M \in \mathbb{R}^{n\times p}$.
	`addmv`	$\beta v + \alpha(A@x)$	Matrix-vector multiply with accumulation. $A \in \mathbb{R}^{n\times m}$, $x \in \mathbb{R}^m$, $v \in \mathbb{R}^n$.
	`addbmm`	$\beta M + \alpha \sum_i (A_i@B_i)$	Batched matrix multiply with reduction over batch dimension and accumulation.
	`baddbmm`	$\beta M_i + \alpha(A_i@B_i)$	Batched matrix multiply with batched accumulation (no reduction).
	`bmm`	$C_i = A_i@B_i$	Batched matrix multiplication. $A \in \mathbb{R}^{b\times n\times m}$, $B \in \mathbb{R}^{b\times m\times p}$.
	`dot`	$\sum_i a_i \cdot b_i$	Dot product of two 1-D tensors $a, b \in \mathbb{R}^n$.
	`addr`	$\beta M + \alpha(u \otimes v)$	Outer product with accumulation. $u \in \mathbb{R}^n$, $v \in \mathbb{R}^m$, $M \in \mathbb{R}^{n\times m}$.
	`linalg_cross`	$a \times b$	Cross product of 3-D vectors, returning vector perpendicular to both inputs.

Table 6: **Comparison between ECS and various RAG baselines in BackendBench**

Operator	Plain LLM	Random	Chunk +Dense	AST +Dense	AST +BM25	AST +Hybrid	ECS (Ours)
acos.default	0.333	0.444	0.444	**0.778**	0.667	0.222	**0.778**
acosh.default	0.333	0.444	**0.778**	0.667	0.667	0.667	**0.778**
addmm.default	**0.333**	0.000	0.000	0.000	0.000	0.000	0.000
addmv.default	0.167	0.000	0.167	0.000	0.000	0.000	**0.222**
addbmm.default	0.000	0.000	0.000	0.000	0.000	0.000	0.000
baddbmm.default	0.000	0.333	0.333	0.333	0.000	0.667	**1.000**
dot.default	0.000	0.000	0.000	0.000	0.000	0.000	0.000
bmm.default	**0.667**	0.333	0.000	0.000	0.000	0.333	0.000
addr.default	0.111	0.167	0.000	**0.222**	0.111	0.111	0.000
asin.default	**1.000**	0.667	**1.000**	0.667	**1.000**	**1.000**	**1.000**
asinh.default	0.333	0.667	**1.000**	**1.000**	**1.000**	0.667	**1.000**
atan.default	0.667	0.667	0.667	**1.000**	0.667	0.667	**1.000**
atan2.default	0.333	0.000	0.296	**0.963**	0.333	0.926	**0.963**
atanh.default	0.667	**1.000**	0.667	**1.000**	**1.000**	**1.000**	**1.000**
ceil.default	0.000	0.000	0.000	0.000	0.000	0.000	**0.333**
linalg_cross.default	**0.222**	0.111	0.000	0.000	**0.222**	0.000	**0.222**
div.Tensor	0.370	0.296	0.296	**0.630**	0.593	0.296	0.333
div.Tensor_mode	0.000	0.000	0.000	0.000	0.000	0.000	0.000
fmod.Tensor	0.593	0.593	0.000	0.000	0.593	**0.630**	0.593
remainder.Tensor	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Over Avg Perf	0.306	0.286	0.282	0.363	0.343	0.359	**0.461**

Table 7: **Transferability of evolved contexts generated by ECS across various models on BackendBench**

Operator	Claude-4.5-Sonnet			DeepSeek-V3.2		
	Plain	AST +Dense	ECS (Ours)	Plain	AST +Dense	ECS (Ours)
acos.default	0.111	0.667	**0.889**	**0.222**	0.111	**0.222**
acosh.default	0.444	**0.778**	0.667	0.000	0.000	**0.222**
addmm.default	0.611	**0.667**	0.389	**0.111**	0.000	0.000
addmv.default	0.222	**0.500**	0.167	0.000	**0.167**	**0.167**
addbmm.default	0.000	0.000	0.000	0.000	0.000	0.000
baddbmm.default	**0.333**	0.167	**0.333**	0.000	0.000	0.000
dot.default	**0.667**	0.333	**0.667**	0.000	0.000	0.000
bmm.default	**1.000**	0.333	0.667	0.000	0.000	0.000
addr.default	**0.389**	0.333	0.333	0.000	0.000	**0.222**
asin.default	0.333	**1.000**	**1.000**	0.000	0.333	**0.667**
asinh.default	0.667	0.667	**1.000**	0.000	**0.333**	0.000
atan.default	0.333	**1.000**	**1.000**	0.000	0.000	**0.667**
atan2.default	0.630	0.926	**0.963**	0.296	0.000	**0.889**
atanh.default	0.667	**1.000**	**1.000**	0.000	0.000	**1.000**
ceil.default	0.333	**0.667**	**0.667**	0.000	0.000	0.000
linalg_cross.default	0.333	0.333	**0.444**	0.000	**0.111**	**0.111**
div.Tensor	**0.778**	0.593	0.370	0.000	0.185	**0.296**
div.Tensor_mode	0.000	**0.593**	0.519	0.000	0.000	0.000
fmod.Tensor	**0.963**	0.296	0.889	0.000	0.000	0.000
remainder.Tensor	0.267	**0.433**	0.267	0.000	**0.067**	0.000
Over Avg Perf	0.454	0.564	**0.611**	0.031	0.065	**0.223**

B.3 Significance Testing

To quantify the stability of our main BackendBench result, we ran each method with n=3 independent seeds and computed Welch's t-test of every baseline against ECS. Table 8 reports the mean correctness, standard deviation, mean difference relative to ECS, t-statistic, degrees of freedom, and two-sided p-value. All baseline comparisons reject at or near $p \approx 0.05$, and the strongest baselines (AST+Dense, AST+Hybrid, AST+BM25) are rejected with $p \in \{0.005, 0.041, 0.042\}$.

Table 8: **Significance testing on BackendBench.** Welch's t-test of each baseline against ECS (n=3). All comparisons reach $p \lesssim 0.05$.

Method	Mean ± SD	Δ mean	t-stat	df	p-value
Plain LLM	0.306 ± 0.069	+0.155	3.62	2.6	0.046
Random	0.286 ± 0.039	+0.175	6.39	3.6	0.005
Chunk + Dense	0.282 ± 0.057	+0.179	4.92	2.9	0.018
AST + Dense	0.363 ± 0.045	+0.098	3.23	3.3	0.042
AST + BM25	0.343 ± 0.026	+0.118	5.50	4.0	0.005
AST + Hybrid	0.359 ± 0.046	+0.102	3.31	3.2	0.041
ECS (Ours)	**0.461 ± 0.027**	—	—	—	—

C τ^2-Bench Details

C.1 Hyperparameter Sensitivity

We study how ECS responds to single-axis perturbations of its main hyperparameters on τ^2-Bench (Pass^1). The default configuration (population size 32, elite fraction 0.6, mutation rate 0.1, full 10-generation evolution) achieves 0.756. Table 9 shows that reasonable perturbations move Pass^1 by at most ∼0.03, indicating that ECS is not finely tuned to one particular setting.

Table 9: **Hyperparameter sensitivity on τ^2-Bench (Pass^1).** Each row perturbs one axis from the default; remaining hyperparameters are held fixed.

Configuration	Pass^1
Default ECS	**0.756**
Population size 32 → 16	0.737
Elite fraction 0.6 → 0.4	0.755
Mutation rate 0.1 → 0.2	0.752
Early stop at iteration 3 (vs. converged)	0.730

The early-stop row also documents the cost–quality trade-off: roughly two thirds of the final gain is recovered after just three generations, consistent with the log-linear convergence predicted by Proposition D.2.

D Convergence Analysis of ECS

We provide a formal convergence analysis that explains the rapid improvement of ECS observed in Section 4. The analysis studies an idealized elitist variant of ECS that isolates the core search mechanism: fixed-size context sets, random replacement mutation, and fitness-guided acceptance. This abstraction omits population-level crossover and LLM-guided refinement, and therefore should be interpreted as a sufficient-condition analysis rather than an exact characterization of the full implementation.

Let $\mathcal{U}$ be a corpus of N context units, $T^\star \subseteq \mathcal{U}$ the optimal target set of size $|T^\star| = r$ maximizing downstream utility, and k the context capacity with $r \leq k < N$. Let $S_t \subseteq \mathcal{U}$ denote the context set at time t, with $|S_t| = k$.

Define the overlap

$$X_t := |S_t \cap T^\star|$$

and the hitting time

$$\tau_q := \inf\{t \geq 0 \mid X_t \geq q\}.$$

We assume excess capacity $\gamma > 0$ such that $k \geq (1+\gamma)r$.

Lemma D.1 (Pure birth process)**.** *Let $U(S) = g(|S \cap T^\star|) + b(S)$, where $g : \{0, \ldots, r\} \to \mathbb{R}$ maps target overlap to base utility and $|b(S)| \leq \epsilon$ is bounded evaluation noise. If $g(i+1) - g(i) \geq \beta > 2\epsilon$ for all $i \in \{0, \ldots, r-1\}$, then under elitist selection the process $\{X_t\}$ is non-decreasing. Consequently, the idealized ECS process is a pure birth Markov process over the states $\{0, \ldots, r\}$.*

Proof. Consider a candidate offspring S' produced from S_t by replacing one context unit. If $|S' \cap T^\star| = X_t + 1$, then

$$U(S') - U(S_t) = \big(g(X_t+1) - g(X_t)\big) + \big(b(S') - b(S_t)\big) \geq \beta - 2\epsilon > 0.$$

Thus the offspring is accepted by elitist selection. If instead $|S' \cap T^\star| = X_t - 1$, then

$$U(S') - U(S_t) \leq -\beta + 2\epsilon < 0,$$

so the offspring is rejected. Overlap-preserving mutations leave X_t unchanged. Therefore $P(X_{t+1} < X_t) = 0$, and the state sequence is non-decreasing. Since transitions depend only on the current overlap state under random replacement mutation, the resulting process is a pure birth Markov process. □

Proposition D.2 (Log-linear convergence)**.** *Under the assumptions of Lemma D.1 and excess capacity $k \geq (1+\gamma)r$ for some constant $\gamma > 0$, the expected time to reach full target overlap satisfies*

$$\mathbb{E}[\tau_r] = \mathcal{O}(N \log r).$$

Proof. At state i, the current set S_t contains i target units and $k - i$ non-target units. The mutation operator draws one outgoing unit $c_{\text{out}} \sim \text{Unif}(S_t)$ and one incoming unit $c_{\text{in}} \sim \text{Unif}(\mathcal{U} \setminus S_t)$, producing

$$S' = (S_t \setminus \{c_{\text{out}}\}) \cup \{c_{\text{in}}\}.$$

The probability of increasing the overlap by one is the probability of removing a non-target unit and adding a missing target unit:

$$t_i = \frac{k-i}{k} \cdot \frac{r-i}{N-k}.$$

By Lemma D.1, the process is pure birth, so the hitting time τ_r can be written as a sum of geometric waiting times W_i for transitions from state i to state $i+1$:

$$\mathbb{E}[\tau_r] = \sum_{i=0}^{r-1} \mathbb{E}[W_i] = \sum_{i=0}^{r-1} \frac{1}{t_i} = \sum_{i=0}^{r-1} \frac{k(N-k)}{(k-i)(r-i)}.$$

Since $\frac{k}{k-i}$ is increasing in i, we bound it by its value at $i = r-1$ and substitute $j = r - i$:

$$\mathbb{E}[\tau_r] \leq (N-k)\frac{k}{k-r+1} \sum_{j=1}^{r} \frac{1}{j}.$$

Using $k \geq (1+\gamma)r$, we have

$$\frac{k}{k-r+1} \leq \frac{(1+\gamma)r}{\gamma r + 1}.$$

Since $H_r = \sum_{j=1}^{r} \frac{1}{j} = \Theta(\log r)$, it follows that

$$\mathbb{E}[\tau_r] \leq (N-k) \cdot \frac{(1+\gamma)r}{\gamma r + 1} \cdot \Theta(\log r) = \mathcal{O}(N \log r).$$

□

This result connects ECS to classical evolutionary computation theory. Elitist evolutionary algorithms over monotone pseudo-Boolean landscapes are commonly analyzed as absorbing Markov chains (Rudolph, 1994), and related results show $\mathcal{O}(N \log N)$ convergence for simple elitist evolutionary algorithms under monotone fitness assumptions (Droste et al., 2002). Our analysis adapts this viewpoint to context search: when useful context units provide strictly positive marginal utility, the overlap with the optimal context set evolves as a pure birth process and reaches the target set in log-linear expected time.

E Cross-Model Transferability Analysis

We complement the convergence result above with a theoretical account of why contexts evolved on one model transfer to another. This formalizes the empirical observation that contexts evolved on Gemini-3-Flash improve Claude-4.5-Sonnet and DeepSeek-V3.2 without re-running the search.

Setup. We adopt the notation from Appendix D. Let $\mathcal{U}$ be a corpus of context units and $T^\star \subseteq \mathcal{U}$ the target set of size r. To reason about transfer, we parameterize the utility function by the evaluating model M:

$$U_M(S) = g_M(|S \cap T^\star|) + b_M(S),$$

where g_M maps target-overlap to base utility under model M, and $b_M(S)$ is bounded nuisance noise with $|b_M(S)| \leq \epsilon_M$.

Definition E.1 (Aligned Marginal Utility)**.** A source model M and target model M' exhibit *aligned marginal utility* with respect to a target set $T^\star$ if:

1. **Strict marginal gain on the source.** g_M satisfies $g_M(i+1) - g_M(i) \geq \beta > 2\epsilon_M$ for all $i \in \{0, \ldots, r-1\}$.
2. **Target monotonicity on the target.** $g_{M'}$ is monotonically non-decreasing in target overlap, and attains its structural maximum when all r target units are present: $g_{M'}(i) \leq g_{M'}(r)$ for all $i \in \{0, \ldots, r\}$.

Intuition. The two models do not need to share absolute performance — only the *direction* of marginal utility. If adding a useful unit helps M, it does not hurt M'. This is far weaker than assuming the models share representations or capabilities; it only assumes that useful structural knowledge is useful in both.

Proposition E.2 (Transferability bound)**.** *Let $C^\star_M$ be the context that ECS evolves on the source model M, and let $C^\star_{M'}$ be the global optimum on the target model M'. Under aligned marginal utility (Definition E.1) and the assumptions of Proposition D.2, the transfer gap is bounded by twice the target model's evaluation noise:*

$$U_{M'}(C^\star_{M'}) - U_{M'}(C^\star_M) \;\leq\; 2\,\epsilon_{M'}.$$

Proof. Because g_M satisfies the strict marginal gain condition, Proposition D.2 guarantees that ECS evolves a context $C^\star_M$ that fully captures the target set: $T^\star \subseteq C^\star_M$, so $|C^\star_M \cap T^\star| = r$.

Evaluating $C^\star_M$ on the target model yields

$$U_{M'}(C^\star_M) \;=\; g_{M'}(r) + b_{M'}(C^\star_M) \;\geq\; g_{M'}(r) - \epsilon_{M'}.$$

For the theoretical optimum $C^\star_{M'}$ on the target model, target monotonicity bounds its structural component by $g_{M'}(r)$, and noise can contribute at most $+\epsilon_{M'}$, giving

$$U_{M'}(C^\star_{M'}) \;\leq\; g_{M'}(r) + \epsilon_{M'}.$$

Subtracting the two bounds yields

$$U_{M'}(C^\star_{M'}) - U_{M'}(C^\star_M) \;\leq\; \big(g_{M'}(r) + \epsilon_{M'}\big) - \big(g_{M'}(r) - \epsilon_{M'}\big) \;=\; 2\,\epsilon_{M'}.$$

□

Remark 1 (Why ECS transfers better than RAG). Aligned marginal utility is easily satisfied when context units encode *structural* knowledge patterns — e.g., the `cute.math.*` API pattern discussed in Section 5.2 — because any sufficiently capable model uniformly benefits from procedural examples. ECS, by optimizing for marginal gain on the source model, systematically isolates these structural units. RAG selects by semantic similarity instead, often retrieving topically related but structurally uninformative content. Because the retrieved set fails to cover $T^\star$, RAG never triggers the $g_{M'}(r)$ peak on the target model.

Remark 2 (End-to-end guarantee). Propositions D.2 and E.2 compose: Proposition D.2 guarantees that ECS discovers $C_M^\star$ in $\mathcal{O}(N \log r)$ expected time; Proposition E.2 guarantees that the discovered context transfers with a penalty bounded only by the target model's evaluation noise.

Remark 3 (Weak-to-strong transfer). When the target M' is more capable than the source M, its structural utility ceiling is at least as high: under the mild assumption $g_{M'}(r) \geq g_M(r)$, Proposition E.2 guarantees that the transferred context $C_M^\star$ reaches a score on M' within $2\epsilon_{M'}$ of M''s own optimum—a score determined by M''s capability, not M's. This is consistent with our empirical observation that contexts evolved on Gemini-3-Flash (0.461 on BackendBench) reach 0.611 when transferred to Claude-4.5-Sonnet without re-evolution.

F Difficulty of Self Supervised Finetuning with Limited Domain Data

We investigate whether supervised fine-tuning (SFT) could improve open-source model performance on CuTeDSL code generation. Our experiments reveal that fine-tuning with limited domain-specific data presents significant challenges.

F.1 Dataset Construction

We curate a training dataset from the CuTeDSL reference documentation, processing 62 Python kernel implementation files and 12 Jupyter notebook tutorials to cover a wide range of GPU architectures. To maximize data diversity, we employ three complementary extraction strategies: full-file extraction (50 samples) using module docstrings as queries, full-notebook extraction (12 samples) converted into interleaved markdown and code, and notebook cell extraction (55 samples) pairing individual code cells with their preceding descriptions. All samples are standardized into a three-turn chat format compatible with SFT frameworks, consisting of an expert CUDA developer system message, a user query synthesized from the source documentation, and the corresponding ground-truth code. The dataset statistics are summarized in Table 10.

Table 10: SFT Dataset Statistics

Statistic	Value
Total training samples	117
Average sequence length	15,940 tokens
Samples truncated (to 14K tokens)	24
Source Python files	62
Source Jupyter notebooks	12
By GPU Architecture	
CUDA (generic)	75
Blackwell	24
Ampere	12
Hopper	3
Multi-GPU/Distributed	3

F.2 Experimental Setup and Result

We fine-tuned the Qwen3-8B model (8.2B parameters) using LoRA (rank=8, α=32) applied to all linear layers, yielding 21.8M trainable parameters (0.27% of the total). To optimize performance, we conducted a learning rate sweep across five values spanning three orders of magnitude, with full training configuration details provided in Table 11.

Table 11: SFT Training Configuration

Hyperparameter	Value
Base Model	Qwen3-8B
Fine-tuning Method	LoRA (rank=8, α=32)
Trainable Parameters	21.8M (0.27%)
Learning Rates	{1e-6, 5e-6, 1e-5, 5e-5, 1e-4}
Global Batch Size	64
Epochs	15
Max Sequence Length	16,384
Precision	bfloat16

Across all learning rates, training failed to converge meaningfully, final loss stagnated at $\sim$5.89 and accuracy remained at baseline levels ($\sim$52%). We observed severe gradient instability driven by the data regime rather than hyperparameter selection, with initial gradient norms exceeding 5×10^7 and values becoming undefined (NaN) by the third epoch. Crucially, this lack of convergence resulted in all checkpoints achieving a zero score on the BackendBench task. These results highlight the fundamental difficulty of fine-tuning on small, specialized datasets with long sequences: with only 117 high-variance examples averaging $\sim$16K tokens, the model lacks sufficient signal to learn generalizable CuTeDSL patterns. Consequently, this motivates our shift to an in-context skill-acquisition approach via ECS, which sidesteps the instability of fine-tuning in this data-scarce regime.