GENERATIVE ADVERSARIAL LOOPS

Anonymous authors
Paper under double-blind review

ABSTRACT

Research progress in AI can be understood as the interaction between two complementary processes: benchmark creation and method discovery. Historically, both processes were driven by human intelligence. However, recent advances in AI have significantly accelerated automated method discovery, while automated benchmark creation has received comparatively less attention. To realize truly self advancing systems, we propose Generative Adversarial Loop (GAL), a generator discriminator framework that alternates between two agentic searches: (1) a discriminator that generates adversarial data to expose weaknesses in current systems, and (2) a generator that discovers algorithms to overcome them. We apply this framework to approximation algorithms for efficient inference. Unlike existing auto research systems, which primarily focus on algorithm discovery, GAL introduces a discriminator agent that automates goalpost setting by continually searching for weaknesses in the current algorithm. We demonstrate adversarial data generation across four tasks: KV compression, sparse video generation, sparse attention, and context extension, where the discriminator identifies weaknesses in state of the art techniques. We further show that GAL enables autonomous improvement for KV compression and context extension, with newly discovered algorithms improving not only on adversarially generated data, but also, in some cases, on established benchmarks. GAL thus provides a path toward autonomous goalpost setting and algorithmic improvement, where AI systems continually discover their own weaknesses and develop methods to overcome them. Code: https://anonymous.4open.science/r/generative-adversarial-loops

1 INTRODUCTION

Research progress in machine learning can broadly be understood as the interaction between two complementary processes: benchmark creation and method discovery. Benchmark creation sets measurable goals that expose limitations in current systems, while method discovery develops algorithms, architectures, and optimization techniques to overcome them. From this perspective, benchmark creation is discriminative: it identifies failure cases, whereas method discovery is generative: it produces solutions. Progress emerges through an alternating cycle between the two: new benchmarks expose capability gaps, enabling focused algorithmic innovation; as methods improve, benchmarks eventually saturate, motivating more challenging evaluations. This cycle has repeatedly shaped machine learning, from MNIST (LeCun et al., 1998), CIFAR (Krizhevsky & Hinton, 2009), and ImageNet (Russakovsky et al., 2015) in image recognition to GLUE (Wang et al., 2018), SuperGLUE (Wang et al., 2019), MMLU (Hendrycks et al., 2020), and GSM8K (Cobbe et al., 2021) in language understanding and reasoning. Sustaining progress therefore requires continuously creating benchmarks that are more challenging, diverse, and discriminative. Yet both sides of this cycle have historically remained largely human driven: researchers design evaluations to expose new weaknesses and develop methods to address them.

Recently, this balance between benchmark creation and method discovery has begun to shift. AI systems increasingly contribute to scientific discovery through rapid prototyping, automated experimentation, large-scale evaluation, and hypothesis generation. Notable successes include AlphaTensor (Fawzi et al., 2022), which discovered improved matrix multiplication algorithms, and AlphaFold (Jumper et al., 2021), which enabled large-scale protein structure prediction. This progress has motivated the development of AI-Driven Research Systems (ADRS) (Cheng et al., 2025). Systems such as AlphaEvolve (Novikov et al., 2025) and OpenEvolve (Sharma, 2025) use iterative

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

search and automated evaluation to discover algorithmic improvements. Broader evolutionary and agentic approaches, such as GEPA (Agrawal et al., 2026), further expand this paradigm. However, existing ADRS approaches automate only one side of the research cycle: method discovery. Nearly all still rely on human-designed benchmarks and evaluation protocols to define progress. As method discovery becomes increasingly automated and scalable, benchmark construction becomes the bottleneck. The next frontier of automated scientific discovery is therefore not only discovering better methods, but also automatically discovering what to evaluate.

This paper addresses this emerging bottleneck by introducing Generative Adversarial Loops (GAL), a framework that automates discriminative benchmark discovery alongside generative method discovery. Unlike the ADRS paradigm, which focuses primarily on the generative loop, GAL automates both processes. Benchmark discovery, however, presents unique challenges. Unlike method discovery, where benchmarks define the objective to optimize, creating meaningful benchmarks often requires reliable ground truth and the ability to identify regions beyond the capabilities of current methods. This dependence has historically made benchmark design a human-driven process. We observe that many important research domains admit ground truth by construction. In such settings, benchmark generation can itself become algorithmic: datasets can be synthesized automatically while preserving reliable supervision, enabling systematic discovery of failure cases and controlled stress-testing of state-of-the-art systems.

One such domain is efficient inference through approximation. To address system bottlenecks in AI inference, many approximation algorithms have been proposed. This work focuses on four such problems: (1) **KV Cache Compression** (Jegou et al., 2024; Liu et al., 2023; Zhang et al., 2023): compressing the KV cache of autoregressive models in embedding space to address the memory bottleneck in LLM inference engines; (2) **Sparse Decoding** (Tang et al., 2024; Hooper et al., 2025; Desai et al., 2026): sparsifying attention computation during decoding to address the memory bandwidth bottleneck of autoregressive decoding; (3) **Sparse Video Generation** (Xi et al., 2025; Yang et al., 2026): sparsifying the prefill workload of diffusion models to enable faster video generation; and (4) **Context Extension** (An et al., 2024; Liu et al., 2025): enabling training-free context extension, allowing models to process contexts substantially longer than their training context window. This domain naturally provides a setting for automated benchmark generation for two reasons. First, benchmarks can often be generated programmatically. For example, long-context benchmarks can be constructed by systematically placing relevant pieces of information at different locations throughout a long context. Second, more importantly, the unapproximated model provides a natural source of ground truth. For example, whether a prompt is suitable for evaluating sparse video generation can be determined by the quality of the video produced by the dense, unapproximated model. These properties enable us to automatically generate benchmarks while retaining reliable supervision for evaluating approximation algorithms.

We formalize benchmark evolution as an iterative process of accumulating data samples, where each iteration discovers a new task on which the state-of-the-art approximation algorithm fails to maintain fidelity to the unapproximated baseline. To discover such failures, we leverage recent autonomous evolution algorithms such as GEPA. When a new failure pattern is identified, we use the ADRS paradigm to evolve the approximation algorithm on the expanded benchmark. Thus, benchmark discovery and method discovery form an alternating loop, with both stages leveraging the same underlying machinery of evolutionary search. More broadly, we argue that automated benchmark generation represents a complementary axis to automated method discovery, which is already gaining momentum, and offers a scalable mechanism for sustaining accelerated research progress.

The focus of this paper is to demonstrate the utility of automated benchmark generation for discovering weaknesses in state-of-the-art algorithms that have seemingly saturated existing public benchmarks and coupling it with algorithm evolution to show possibility of complete autonomous progress. To this end, we show that GALs can uncover weaknesses in four algorithms: FastKVZip (Kim et al., 2026a) for KV cache compression, PQCache (Zhang et al., 2025) for sparse attention, SVG (Xi et al., 2025) for sparse video generation, and DCA (An et al., 2024) for context extension. Most of the adversarial failure modes uncovered by GALs are interpretable, providing insight into the limitations of existing approximation algorithms. We demonstrate the generative leg of the GAL loop through two rounds of algorithmic improvement for Context Extension and KV compression. In both cases, the newly discovered algorithms in each iteration of GAL, address the failure modes identified by the newly generated benchmarks while also maintaining, and often improving, performance on previously established benchmarks. This behavior illustrates the feedback loop: newly generated

benchmarks expose weaknesses in existing methods, which motivates the discovery of improved methods that address these weaknesses without sacrificing performance on existing evaluations.

2 Related Work

Auto research and evolution algorithms Recent advances in auto research have increasingly used evolutionary search to create autonomous loops that iteratively propose, evaluate, and refine solutions against fixed benchmarks or metrics (Karpathy, accessed 2026). In this paradigm, an LLM is not treated as a one shot solution generator, but as a search operator that proposes mutations or new candidates, while automated evaluation provides the objective signal needed to guide subsequent iterations. AlphaEvolve (Novikov et al., 2025) exemplifies this approach by combining LLM generated program mutations with evolutionary selection, allowing promising programs to be iteratively improved and complementary ideas to be explored across a population of candidates. GEPA (Agrawal et al., 2026), in contrast, uses LLMs to reflect on execution traces and failures to generate targeted mutations, while maintaining a Pareto frontier of candidates that perform well on different subsets of the evaluation data. Related efforts such as autoresearch further demonstrate the potential of autonomous experimentation, where agents repeatedly modify a system, run experiments, and retain improvements based on measured performance. AI Driven Systems Research (Cheng et al., 2025) provides a large scale case study of using such automated research agents to tackle challenging systems problems. Collectively, these efforts establish a paradigm in which automated evaluation and evolutionary proposal mechanisms enable LLMs to autonomously navigate large spaces of possible solutions. However, existing auto research systems have primarily used these capabilities to improve a solution against a fixed task, benchmark, or metric. In contrast, we repurpose the same evolutionary machinery to generate adversarial data that targets both the task and the current best solution, enabling the automated discovery of failure modes that can subsequently be used to improve the underlying system.

Synthetic data generation The field of synthetic data generation is closely related. One of the early influential approaches, Self-Instruct (Wang et al., 2023), showed that an LLM can generate its own instructions, inputs, and outputs, and use the resulting synthetic dataset for instruction tuning. This led to various efforts to evolve the difficulty of synthetic examples and adapt them to different domains (Xu et al., 2024; Luo et al., 2024; 2025). Additionally, LLMs have been used to improve the quality of generated data through critique and evaluation (Cui et al., 2023; Madaan et al., 2023). A particularly important development has been the use of synthetic data for reasoning distillation. DeepSeek-R1 (Guo et al., 2025) demonstrates that a strong reasoning model can generate large numbers of reasoning trajectories that are subsequently used to train smaller models. Collectively, these approaches treat synthetic data generation as a mechanism for constructing increasingly useful training distributions, with generation, evaluation, filtering, and distillation forming a recurring loop. Our work differs in that we use automated search not primarily to generate useful training examples, but to actively generate adversarial examples that target weaknesses of the current best solution.

Other related works There have been several recent and concurrent works that target dataset evolution. BenchEvolver(Wu et al., 2026) proposes a system for evolving coding benchmarks to generate challenging datasets for frontier models. Data-Prompt Co-evolution(Lee & Kahng, 2026) evolves test sets and prompts to develop safer and more policy compliant systems. Hacker-Fixer Loops(Zhong et al., 2026) evolve agentic benchmarks to mitigate reward hacking and ensure that improvements reflect genuine progress. These works are aligned with our approach, with the differences being their areas of application and the domain specific validation used to assess the generated data.

3 Generative Adversarial Loops

In this section, we describe the general formulation of a Generative Adversarial Loop (GAL). Consider a system $\mathcal{S}$, such as an inference system, that takes a sequence of tokens (e.g., text, images, or other modalities) as input and produces a sequence as output. The system is parameterized by an artifact A, such as an efficiency algorithm, along with other parameters that remain fixed throughout the lifetime of the loop. Let $\mathcal{D}$ denote the set of all valid inputs (or data points), and let $\mathcal{A}$ denote the set of all valid artifacts that can parameterize $\mathcal{S}$. Note that the set $\mathcal{D}$ is enumerable, and in systems of interest, such as LLM inference systems, it is also finite, since the system can only ingest finite-length strings. If N is the maximum length, in tokens, of a string the system can ingest, then D_N, the set of valid inputs of length $<= N$, is actually a finite set. Let $\mathcal{E}$ be an evaluation function that determines

whether the response of $\mathcal{S}(A)$ on a given data point is successful. Formally, $\mathcal{E} : \mathcal{A} \times \mathcal{D} \to \{0, 1\}$, where $\mathcal{E}(A, d) = 1$ indicates that the system $\mathcal{S}(A)$ successfully solves data point $d \in \mathcal{D}$, while $\mathcal{E}(A, d) = 0$ indicates failure. For an artifact $A \in \mathcal{A}$, we define its coverage over the task space as

$$\mathcal{C}(A) = \{d \in \mathcal{D} \mid \mathcal{E}(A, d) = 1\}. \tag{1}$$

Thus, $\mathcal{C}(A)$ represents the subset of the task space on which the artifact A successfully performs the task. In practice, evaluation is performed on a finite benchmark $B \subseteq \mathcal{D}$, consisting of a representative set of data points used to measure system performance. For example, B may comprise long-context benchmarks Hsieh et al. (2024); Bai et al. (2024); Li et al. (2023); Lee et al. (2024) for KV-cache compression and sparse attention, or prompts from vBench (Huang et al., 2024; Zheng et al., 2025) for sparse video generation. We define the benchmark-level evaluation of an artifact as $\mathcal{E}(A, B) = |\{d \in B \mid \mathcal{E}(A, d) = 1\}|/|B|$. This quantity measures the fraction of benchmark instances solved by the system parameterized by A. We want to find an algorithm $A^* \in \mathcal{A}$ such that

$$A^* \in \arg\max_{A \in \mathcal{A}} |\mathcal{C}(A)| \tag{2}$$

3.1 Discriminator and Generator Agents

Assume we start with a benchmark B_0 and an artifact A_0 s.t. $\mathcal{S}(A)$ solves the benchmark B_0, i.e., $\mathcal{E}(A_0, B_0) = 1$. We decompose the process of artifact discovery into two phases, executed alternately. In the general case of ith iteration, we have B_{i-1}, A_{i-1} where $\mathcal{E}(A_{i-1}, B_{i-1}) = 1$

Discriminator phase. In the ith iteration of this phase, we seek a data point b on which A_{i-1} fails:

$$\textbf{find } b \quad \text{s.t.} \quad b \in \mathcal{D} \setminus \mathcal{C}(A_{i-1}) \tag{3}$$

Note that, in general, $\mathcal{C}(A_{i-1})$ can be larger than the benchmark B_{i-1} used to evaluate A_{i-1}. The discriminator phase seeks to explore the task space beyond the current benchmark by discovering an example that is difficult for the current system $\mathcal{S}(A_{i-1})$ to solve. Let $b \in \mathcal{D} \setminus \mathcal{C}(A_{i-1})$ denote such an example. We append this newly discovered example to the existing benchmark B_{i-1} to obtain an expanded benchmark $B_i = B_{i-1} \cup \{b\}$, $B_0 \subset B_1 \subset B_2 \subset \ldots \subseteq \mathcal{D}$. As the loop proceeds, the benchmark accumulates increasingly challenging examples. In practice, we generalize this process to discover a batch of data points on which $\mathcal{S}(A_{i-1})$ fails within each discriminator phase.

Generator phase. This phase seeks an artifact A_i that solves the expanded benchmark B_i:

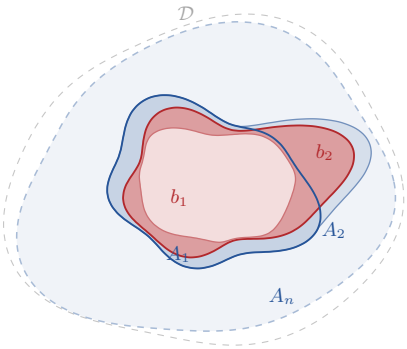
$$\textbf{find } A_i \quad \text{s.t.} \quad \mathcal{E}(A_i, B_i) = 1. \tag{4}$$

This phase therefore seeks a new artifact that addresses the newly discovered failure cases while preserving correctness on previously discovered examples. In this sense, the generator phase corresponds to the search for improved algorithms or systems, which has traditionally been the focus of evolutionary algorithms and automated research systems. The formulation above presents a simplified, deterministic view in which the correctness of $\mathcal{S}(A)$ on each data point is binary. The framework can naturally be extended to stochastic systems, such as LLM inference engines, by treating the evaluation outcome as a random variable. Specifically, for a given artifact A and data point d, we can define $\mathcal{E}(A, d) \sim \text{Bernoulli}(p_{A,d})$, $\mathbb{E}[\mathcal{E}(A, B)] = \sum_{d \in B} p_{A,d}/|B|$, where $p_{A,d}$ denotes the probability that $\mathcal{S}(A)$ successfully solves d. The benchmark-level evaluation can then be defined in terms of the expected success rate $\mathbb{E}[\mathcal{E}(A, B)]$.

Since the task space $\mathcal{D}_N$ is finite, it is guaranteed that the sequence of expanding benchmarks $\{B_i\}$ will eventually reach $\mathcal{D}_N$. Thus, if at every step, our generator is able to find an artifact, we are guaranteed to converge to the true artifact A^* if one exists. This process is illustrated in Figure 1.

3.2 Evolution in practice

Since we apply Generative Adversarial Loops to the specific domain of efficiency problems in LLM inference, we have a domain-specific mechanism for both generating candidate examples and verifying their correctness. For long-context problems such as Sparse Attention, KV Compression, and Context Extension, we find it useful to iterate on a program that generates examples rather than directly optimizing the examples themselves. First, this leads to interpretable failure patterns once we identify a candidate point b, which would otherwise be difficult to decipher from long-context example. Second,



Algorithm 1 Local Optimization

Require: initial algorithm A_0, initial benchmark $B_0 \subset \mathcal{D}$

for $i = 1, 2, \ldots, n$ **do**
  $b \leftarrow$ **find** b s.t. $b \in \mathcal{D} \setminus \mathcal{C}(A_{i-1})$
  $B_i \leftarrow B_{i-1} \cup \{b\}$
  $A_i \leftarrow$ find $\mathcal{A}$ s.t. $\mathcal{E}(A_i, B_i) = 1$
end for
return A_n

Figure 1: Visualization and the iterative algorithm. The red area represents the benchmark set and blue area is the artifact coverage. After the first iteration $\mathcal{C}(A_1)$ contains b_1, so that benchmark is solved. The Discriminator then extends the benchmark with b_2 to form B_1, which is no longer contained in $\mathcal{C}(A_1)$. The Generator then answers with A_2 which covers the new set. After enough number of iterations, the cover of algorithm will approach the dataset $\mathcal{D}_{\mathcal{N}}$.

it provides a guarantee of validity by construction. Since the required information is first generated and subsequently embedded into a long context, we can guarantee that the answer is present in the context.

We use a second mechanism to further ensure dataset validity: we evaluate each candidate using unapproximated model inference and require it to achieve good performance on the proposed example. Thus, we modify the definition of the discriminator as

$$\textbf{find} \quad b \in \mathcal{D} \quad \text{s.t.} \ \mathbb{E}(\mathcal{E}(\phi, b)) > \tau_0 \wedge (\mathbb{E}(\mathcal{E}(\phi, b)) - \mathbb{E}(\mathcal{E}(A_{i-1}, b))) > \tau$$

where $\mathcal{E}(\phi, b)$, i.e., $A = \phi$, refers to unapproximated LLM inference. Here, τ_0 specifies the minimum performance required from unapproximated inference. This ensures that we do not penalize approximation algorithms on tasks that the unapproximated baseline itself cannot solve, and we then search for data points on which the approximated inference performs worse than the unapproximated inference by at least a gap in quality of τ.

Both the discriminator and generator are search problems, which we propose to solve using approaches in auto-research. To this end, we can use a range of algorithms, including but not limited to GEPA, OpenEvolve, Cursor, and Claude Agents. For large projects, where changes to a single file may depend on many other files across the codebase, Cursor and Claude coding harnesses are effective at providing LLMs with the relevant context. This makes them effective during the generative phase, where successful modifications often require understanding and coordinating changes across multiple parts of a project. We denote a proposer with P and a proposed candidate $c_{t+1} = P(c_t, \mathcal{E}_t, \mathcal{F}_t)$, where c_t is the candidate being searched over at time step t and $\mathcal{F}_t$ is the feedback received via some evaluator. We iterate in a loop multiple times until the proposer finds a valid next candidate.

Discriminator and Generator At each iteration i, given the current benchmark set B_{i-1} on which algorithm A_{i-1} succeeds, Discriminator uses proposer P to generate a new data point $b = P(B_{i-1}, \mathcal{E}_{i-1}, \mathcal{F}_{i-1})$, and evaluate A_{i-1} on b. We keep generating a data point till we find one that A_{i-1} cannot solve. This b is then appended to B_{i-1} to get B_i Then, Generator takes over and uses proposer P to generate a new algorithm A_i which not only solves b but also solves B_{i-1}.

4 Experiments on Efficiency Problems in AI Inference

In this section, we describe the problem domains we target, the algorithms we use to represent the state of the art in each domain, and representative adversarial examples discovered by the GAL discriminator, along with their interpretation.

Sparse Video Generation

State-of-the-art video generation typically relies on diffusion models, which iteratively refine a video over multiple steps. Each step requires computation over the full sequence of video tokens, making generation increasingly expensive as resolution and duration grow. As a result, even a few

Figure 2 Sparse Attention via SVG

1. Adversarial prompt, validation, and results

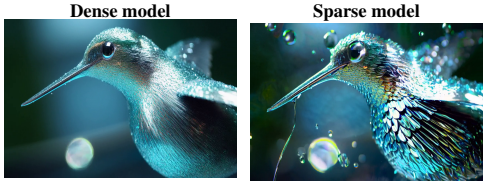
Extreme macro **slow-motion hovering hummingbird** from liquid gallium… **dense fractal metallic feathers**, **razor-thin highlights**; wings blur… molten beads, frost, pinpoint reflections…

Validation. The dense model produces a good video.

Results.

	PSNR ↑	SSIM ↑	LPIPS ↓
Score	15.57	0.60	0.45

2. Dense and sparse video snapshots



Explanation. The prompt (full prompt in appendix) is very detailed, requiring high-resolution video and computation and complex light and mirrored surface interaction. **Settings.** We use the Wan2.1-T2V-1.3B model for 81 frames at 720 × 1280 pixels. The sparse version uses 30% sparsity; all other settings follow the SVG recommendations. Dense and sparse models use the same seed and sampling schedule.

Figure 3 Context Extension via DCA

1. Adversary, validation, and results

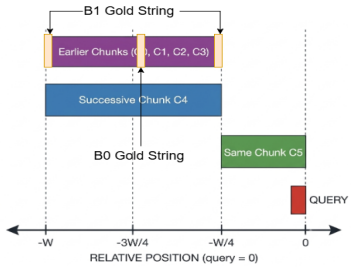
```
def build_adversary():
 s = make_goldstr(); on_chunk_boundary(s)
 add_distractors(); return instance
```

Validation: 1. **By construction:** create Q/A before placing the target. 2. **Baseline control:** solved at native length W.

Results.

	$\mathcal{C}@N$	$\mathcal{A}@\lambda = 4 \times N$	Gap
B_0	1.00	0.90	0.10
B_1	0.95	0.50	0.45

2. DCA position mapping



Explanation. The evolved generator moves the target to a DCA chunk boundary and adds same-format distractors. Position remapping can fragment the target in the query's effective view, while locally coherent distractors remain competitive. **Settings.** We use Llama3-8B-Instruct with an 8K native context window and 4× context extension.

seconds of high-resolution video can take tens of minutes to generate on high-end GPUs. *Sparse prefill* addresses this by reducing attention computation while preserving generation quality. One state-of-the-art approach is SVG (Sparse Video Generation) (Xi et al., 2025). SVG identifies two distinct attention patterns in video - *spatial* and *temporal* and applies different sparsity patterns to each head type. It substantially reduces computation while maintaining quality close to dense generation, reporting PSNR values of approximately 28–29, SSIM around 0.9, and LPIPS around 0.1–0.12, with comparable or sometimes better image quality and temporal consistency.

Example Adversary: GAL's discriminator identified adversarial prompts that cause SVG to deviate substantially from dense video generation. One such example is shown in Figure 2. For this prompt, the relative PSNR and SSIM are substantially lower, while LPIPS is higher, indicating a larger deviation from the video produced by the dense model. The adversarial prompt requests a high-resolution video involving complex interactions of light, creating a challenging generation setting. Such prompts appear to expose a weakness in the sparsity patterns used by SVG: attention that can be safely omitted for typical video-generation prompts may become important when the scene contains fine-grained spatial structure and complex temporal interactions. Thus, while SVG performs well on standard benchmarks, the adversarial example demonstrates that its approximation can breakdown under specific, systematically generated inputs.

Context Extension

Extending a model's context window is computationally expensive, motivating training-free methods that extend the effective context beyond the native limit. Dual Chunk Attention (DCA) (An et al., 2024) is a state-of-the-art approach, natively supported in vLLM (Kwon et al., 2023), with particularly strong results on Qwen models. DCA addresses out-of-distribution relative positions by remapping token positions based on their distance from the query. As illustrated in Figure 3, tokens from distant chunks are mapped to a shared local range 0 to C, while nearby chunks are positioned immediately

Figure 4 KV Compression via FastKVZip

1. Adversary, validation, and results

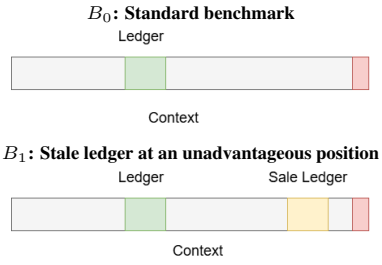
```
def build_adversary():
  fresh, stale = make_ledgers()
  place(fresh, mid); place(stale, ↪
    near_end)
  return instance
```

Validation: 1. **By construction:** a solution exists. 2. **Baseline control:** the model with full KV performs well.

Results.

	Accuracy
Uncompressed	0.93
FastKVZip	0.35
Gap	**0.58**

2. Standard and evolved benchmarks



Explanation. Stale and fresh information is harder to distinguish during compression. To make retrieval harder, the stale ledger is placed near the prompt and the fresh ledger farther away. **Settings.** We use FastKVZip with Qwen3-8B, a 32K-token context, and $4\times$ KV-cache compression.

Figure 5 Sparse Decode Attention via PQCache

1. Adversary, validation, and results

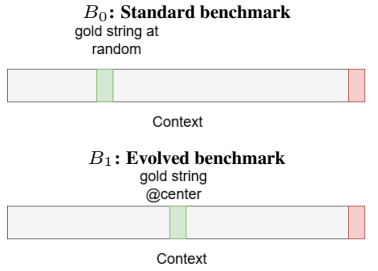
```
def build_adversary():
  s = make_gold_str(); place(s, center)
  return instance
```

Validation: 1. **By construction:** a solution exists. 2. **Baseline control:** dense attention retrieves the gold string.

Results.

	Accuracy
Dense attention	0.80
PQCache (2%)	0.00
Gap	**0.80**

2. Standard and evolved benchmarks



Explanation. Attention scores are typically high near the extremities and low in the middle. Extensive training largely mitigates this lost-in-the-middle effect for dense attention, but it resurfaces under top-k sparse attention such as PQCache. **Settings.** We use Qwen3-4B-Instruct-2507 with PQCache at 2% sparsity.

preceding the query, where C is the chunk size. DCA typically enables $2\times$–$4\times$ context extension while preserving perplexity and retrieval performance, with some results extending beyond this range.

Example Adversary: GAL's discriminator identified adversarial patterns that cause DCA to fail substantially. One such pattern is shown in Figure 3. The discriminator generates a program parameterized by the context length. At the native context length, the base model successfully solves the problem. However, when the same problem is extended to $4\times$ the native context window, the DCA-enabled model fails. The adversarial pattern exploits the interaction between DCA's positional remapping and the location of the retrieval information. Specifically, the relevant retrieval information is placed at the boundary of a chunk. Although DCA preserves relative positioning locally within each chunk, its remapping can cause tokens belonging to the same retrieval pattern to be scattered across different relative positions when the information lies near a chunk boundary. For a distant query, this scattering changes the positional structure of the retrieval information relative to the query. As a result, the model may fail to recognize and retrieve the relevant information, despite the information itself being present in the context.

KV Compression

Long-context autoregressive models face a memory bottleneck in the form of the KV cache, which stores per-token embeddings to reduce generation-time computation. Since its size grows linearly with context length, large KV caches can limit batch sizes or require CPU offloading, reducing serving throughput. KV Compression addresses this by reducing the cache's memory footprint. KVZip (Kim et al., 2026b) and its lightweight variant FastKVZip (Kim et al., 2026a) are state-of-the-art approaches that retain a small subset of important keys and values to approximate the full context. They achieve up to $4\times$ compression while maintaining performance on retrieval, contextual QA, and related tasks.

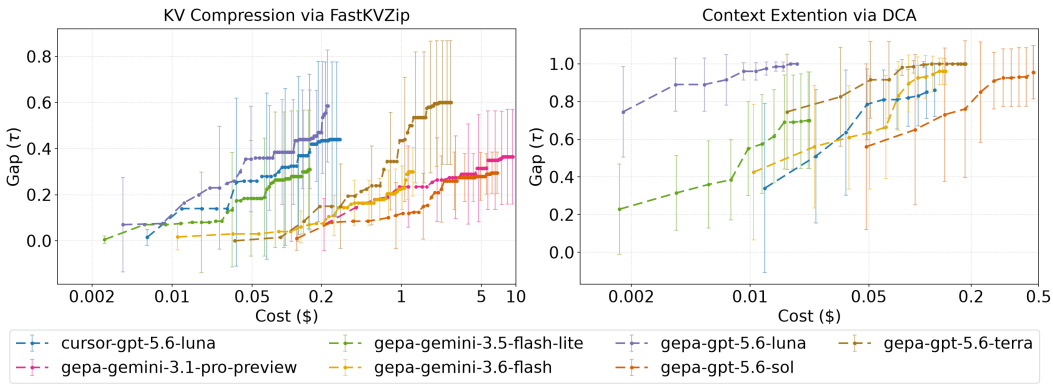


Figure 6: Best gap (τ) versus cost for KV compression (FastKVZip, left) and context extension (DCA, right). Gap is averaged over 10 runs per setting.

Example Adversary: GAL's discriminator identified adversarial patterns that cause FastKVZip to fail substantially. One such pattern is shown in Figure 4. As we can see, FastKVZip fails significantly, dropping the accuracy score from 0.93 for the uncompressed model to 0.35 for the compressed model. The adversarial pattern exploits compression errors by constructing a fresh ledger and a stale ledger of key-value pairs. The fresh ledger is placed mid-context, while the stale ledger appears at the end with each entry explicitly marked stale. The query specifically asks for the fresh ledger. When the context comprises news or articles that are time sensitive, this can be a practical occurrence where stale information is also present in the context.

SPARSE DECODING

Autoregressive LLM decoding is memory-bound, primarily due to reading the large KV cache from GPU memory, limiting token throughput. Sparse attention reduces this memory movement by attending to a subset of tokens, making token selection critical. PQCache (Zhang et al., 2025) is a state-of-the-art sparse attention method that selects relevant tokens using a compact product-quantized representation of the KV cache. It achieves top performance on sparse attention leaderboard for Qwen models (Desai et al., 2025). We therefore use Qwen3-4B-Instruct-2507 with PQCache at 2% sparsity.

Example Adversary: GAL's discriminator identifies adversarial patterns that cause PQCache to fail substantially. One such pattern is shown in Figure 5. As we can see, PQCache performs significantly worse on this pattern. The adversarial pattern exploits the fact that the nature of most attention is such that it naturally pays more attention to sink tokens (i.e., at the start of the context) and local tokens (near the query). When important keys are embedded in the middle, the lost in the middle phenomenon is observed. Newer models, such as Qwen, after training for long contexts, have largely solved the lost in the middle problem, but when sparsified, the phenomenon shows up again. At higher sparsities (e.g., 5% or more), this pattern is not adversarial to PQCache.

COST OF DISCRIMINATOR REQUIRED TO FIND b

Figure 6 shows the cost (in $) required to find adversarial data points b for state-of-the-art models from the Gemini and GPT families. We use GEPA and Cursor as the evolution harnesses. Across models, the evolved solutions consistently increase the gap between unapproximated and approximated inference, suggesting that the discriminator recipe transfers across providers and model capabilities. Surprisingly, lighter models such as Gemini Flash Lite, Gemini Flash, and GPT Luna are substantially more effective at generating adversarial data than larger models such as GPT Sol and Gemini Pro. This difference is even more pronounced in cost: lighter models can discover adversarial data for a fraction of a dollar, while larger models often struggle under the same instructions.

Cost Certificate. The cost required to break an algorithm can serve as a certificate of robustness. Approximate inference algorithms may deviate from unapproximated inference on some inputs, but the cost of discovering such failures provides a quantitative measure of how difficult they are to expose under a specified adversarial generation and evaluation procedure. We refer to this measure as the *cost certificate*. For example, Figure 6 shows that FastKVZip can be broken with GPT Luna (GEPA) for $0.20 on average at a gap of 0.5, while DCA can be broken for less than $0.01. The certificate also enables comparisons within a domain: at 2% sparsity, PQCache can be broken for less than $0.01, whereas breaking Oracle Topk requires $5.06.

	Evolved benchmarks		Paper benchmarks (RULER-4K hard)					
algorithm	B_1	B_2	**qa_1**	**qa_2**	**vt**	**cwe**	**fwe**	**niah_mv**
no-press	1.00	0.95	0.84	0.60	1.00	0.962	0.8867	0.995
Compactor	0.25	0.45	0.54	0.48	1.00	0.76	0.80	0.92
Gen-1	0.85	0.25	0.62	0.54	1.00	0.736	0.7933	0.92
Gen-2	0.95	1.00	0.54	0.48	1.00	0.728	0.8133	0.985

	Evolved benchmarks			Paper benchmarks							
				passkey		needle		PG19 perplexity		Coursera	ScienceFiction
algorithm	**seed**	B_1	B_2	**16k**	**32k**	**16k**	**32k**	**16k**	**32k**	∼9.9k	∼15.2k
Full	0.00	0.00	0.00	0.00	0.00	0.00	0.00	134.07	1025.96	40.0	50.0
DCA	0.90	0.00	0.40	1.00	1.00	1.00	1.00	12.51	12.45	58.0	68.0
Gen-1	1.00	1.00	0.57	1.00	1.00	1.00	1.00	12.39	12.19	60.0	70.0
Gen-2	1.00	0.97	0.83	1.00	1.00	1.00	1.00	12.36	12.12	58.0	74.0

Figure 7: **GAL on KV Compression (top) and Context Extension(Bottom)** Compactor Settings: $4\times$ compression, model= Qwen3-4B-Instruct-2507 using Cursor evolution harness. DCA Settings: $4\times$ extension with model=Llama3 with 8K context. The generated algorithms in both cases perform well on adversarial benchmarks while maintaining, and in some cases improving, performance on standard benchmarks.

Closing the Loop

We run GAL with both a discriminator, implemented using GEPA evolution, and a generator, implemented using Cursor/Claude evolution. We run two complete GAL iterations on two problem domains: context extension, using DCA as the state-of-the-art baseline, and KV compression, using Compactor (Chari & Van Durme, 2025). The results are presented in Figure 7. In both domains, GAL discovers algorithms that not only improve performance on the adversarial data generated by the discriminator, but also improve over the standard benchmarks that constitute B_0.

KV Compression. We use Qwen3-4B-Instruct-2507 at $4\times$ compression. In round 1, the discriminator hid a ledger needle among many lookalike rows in the middle of the context, causing Compactor's global top-k selection to discard the relevant ledger entry. The generator countered by introducing a stratified budget that allocates compression capacity across local chunks, ensuring that all parts of the context receive priority. In round 2, the discriminator constructed a multi-hop chain using sparse route IDs that the updated Gen-1 algorithm still dropped. The generator countered by combining KeyDiff (Park et al., 2026), which preserves distinctive keys, with attention protection and value-norm signals, allowing both hop IDs and semantic content to survive under a unified scoring function.

Context Extension. We use Llama3-8B-Instruct with $4\times$ context extension. In round 1, the discriminator constructed a needle-in-a-haystack task in which the target needle appeared in the middle of the context among lookalike distractors, exposing a failure mode of DCA described in Figure 3. The generator countered by incorporating ideas from Self-Extend, a recent approach to context extension. In round 2, the discriminator filled the context with many near-identical records, making the task more challenging under DCA's position adjustments. The generator countered by introducing continuous positioning and preserving the original value scale to better fit the extended context within the model's trained window.

5 Conclusion

In this work, we present Generative Adversarial Loops (GALs), a framework for automated algorithm discovery through the interaction of generation and discrimination. While prior work has largely automated the generative component, we argue that the discriminator is equally important and underexplored. We instantiate GALs using GEPA, Cursor, and Claude Agents on approximation problems in efficient AI inference. GALs uncover interpretable failure patterns in existing algorithms and, when coupled with the generative loop, use these failures to discover improved algorithms that generalize across diverse benchmarks. More broadly, GALs can become a useful tool for researchers, accelerating algorithmic iteration while providing a cost certificate for robustness: the adversarial effort or computational cost required to expose a failure mode. While this certificate does not establish universal robustness, it quantitatively measures how difficult a method is to break under a specified adversarial generation and evaluation procedure.

AI USE STATEMENT

We used AI to polish the writing of this paper. The content, ideas, arguments are owned by the authors.

ETHICS STATEMENT

Our work focuses on automating the process of making AI efficient. We do not identify any ethical considerations arising specifically from this work

REPRODUCIBILITY STATEMENT

We release the code used to generate the results reported in this paper. Due to the inherent randomness of LLM responses, the reported numbers may vary slightly across runs. The reported cost numbers are averaged over 10 runs, with the corresponding standard deviations indicated.

REFERENCES

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. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Chenxin An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. Training-free long-context scaling of large language models. *arXiv preprint arXiv:2402.17463*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172.

Vivek Chari and Benjamin Van Durme. Compactor: Calibrated query-agnostic kv cache compression with approximate leverage scores. *arXiv preprint arXiv:2507.08143*, 2025.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, et al. Barbarians at the gate: How ai is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. *arXiv preprint arXiv:2310.01377*, 2023.

Aditya Desai, Kumar Krishna Agrawal, Luis Schroeder, Prithvi Dixit, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Introducing sky-light: Advancing the frontier of sparse attention research. November 2025. URL https://sky-light.eecs.berkeley.edu/.

Aditya Desai, Kumar Krishna Agrawal, Shuo Yang, Alejandro Cuadron, Luis Gaspar Schroeder, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. vAttention: Verified Sparse Attention via Sampling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zzTDulLys0.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed Attention: Accelerating Long Context Length LLM Inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21807–21818. IEEE, 2024.

Simon Jegou, Maximilian Jeblick, Alessio Devoto, Jiwei Liu, and David Austin. Kvpress: Efficient kv cache compression for long-context llms, 2024. URL `https://github.com/NVIDIA/kvpress`. Version 1.2.0.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Andrej Karpathy. autoresearch: https://github.com/karpathy/autoresearch, accessed 2026. GitHub repository.

Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*, 2026a.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *Advances in Neural Information Processing Systems*, 38:167563–167591, 2026b.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Jinhyuk Lee, Anthony Chen, Zhuyun Dai, Dheeru Dua, Devendra Singh Sachan, Michael Boratko, Yi Luan, Sébastien M. R. Arnold, Vincent Perot, Siddharth Dalmia, Hexiang Hu, Xudong Lin, Panupong Pasupat, Aida Amini, Jeremy R. Cole, Sebastian Riedel, Iftekhar Naim, Ming-Wei Chang, and Kelvin Guu. Can Long-Context Language Models Subsume Retrieval, RAG, SQL, and More? *arXiv preprint arXiv:2406.13121*, 2024.

Minjae Lee and Minsuk Kahng. Data-prompt co-evolution: Growing test sets to refine llm behavior. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pp. 1–17, 2026.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? *arXiv preprint arXiv:2311.04939*, 2023.

Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Qipeng Guo, Yuerong Song, Kai Lv, Hang Yan, Linlin Li, Qun Liu, and Xipeng Qiu. Reattention: Training-free infinite context with finite attention scope. In *International Conference on Learning Representations*, volume 2025, pp. 95458–95478, 2025.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, et al. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *International Conference on Learning Representations*, volume 2025, pp. 49573–49609, 2025.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *International Conference on Learning Representations*, volume 2024, pp. 27168–27188, 2024.

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. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments. *Advances in Neural Information Processing Systems*, 38:5983–6019, 2026.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *International Conference on Machine Learning*, 2024.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pp. 353–355, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Yangzhen Wu, Aaron J Li, Wenjie Ma, Li Cao, Ziheng Zhou, Mert Cemri, Shu Liu, Yuran Xiu, Chenxiao Yan, Haikun Zhao, et al. Benchevolver: Frontier task synthesis via solution-centric evolution. *arXiv preprint arXiv:2606.01286*, 2026.

Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, et al. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pp. 30745–30766, 2024.

Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Chenfeng Xu, Kelly Peng, et al. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. *Advances in Neural Information Processing Systems*, 38:96965–96991, 2026.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. PQCache: Product Quantization-based KVCache for Long Context LLM Inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025. URL https://arxiv.org/abs/2407.12820.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=RkRrPp7GKO.

Dian Zheng, Ziqi Huang, Hongbo Liu, Kai Zou, Yinan He, Fan Zhang, Lulu Gu, Yuanhan Zhang, Jingwen He, Wei-Shi Zheng, et al. Vbench-2.0: Advancing video generation benchmark suite for intrinsic faithfulness. *arXiv preprint arXiv:2503.21755*, 2025.

Ziqian Zhong, Ivgeni Segal, Ivan Bercovich, Shashwat Saxena, Kexun Zhang, and Aditi Raghunathan. Hardening agent benchmarks with adversarial hacker-fixer loops. *arXiv preprint arXiv:2606.08960*, 2026.

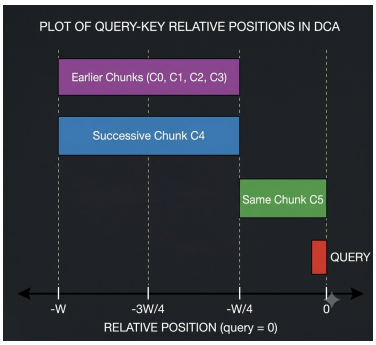


Figure 8: DCA positional remapping across chunks. The chunk size is $3W/4$, and the query is located at position $4W$.

A Appendix

A.1 Task: Context Extension

Description. Context extension asks whether a model trained with context window W can process inputs substantially longer than W. We study Dual Chunk Attention (DCA), a training-free approach that extends a pretrained model's usable context without modifying its weights. DCA partitions a long input into chunks and remaps token positions using inter-chunk, successive, or intra-chunk schemes so that each query–key offset remains within the model's pretrained positional range. For a query, tokens in its own and immediately preceding chunks remain nearby, while tokens in earlier chunks are mapped near position zero; Figure 8 illustrates this construction.

Hypothesis. Prior DCA results suggest that it can extend context length by $4\times$–$8\times$ with limited quality degradation; we test whether an adversarially constructed retrieval task exposes a larger degradation.

Setup. Because manually generating and interpreting long-context instances is impractical, the discriminator produces a Python program that generates retrieval problems rather than producing the problems directly. Figure 9 shows one evolved generator: it controls the needle representation, identifier structure, placement depth, question type, and distractors while remaining blind to the tested context length.

Dataset validity. Each example has a solution by construction: the generator creates the question and answer before placing the associated record in the context. During optimization, we also require strong native-window performance at length W. This constraint rules out intrinsically unsolvable instances, so an observed gap isolates degradation caused by context extension rather than baseline model failure.

Discovered DCA adversary. Figure 10 shows an evolved pattern with chunk size W. The generator places the target record at a chunk boundary and distributes same-format distractors throughout the context. Under DCA's position remapping, the target's tokens can become separated in the query's effective positional view, disrupting the contiguous retrieval pattern encountered during pretraining. Distractors that remain locally coherent can then compete with the fragmented target, increasing the likelihood of retrieving an incorrect record.

B GEPA Setup

B.1 Purpose and Scope

This appendix describes how ADVERSEARCH-AGENT uses GEPA Agrawal et al. (2026) as the outer optimization loop for discovering adversarial candidates. GEPA iteratively mutates a *candidate artifact* under a fixed, harness-defined evaluation protocol. Depending on the task, this artifact is either (i) a *code candidate*, a function that constructs adversarial inputs or evaluation examples, or

```python
def generate_examples(num_examples, rng):
    rows = []
    for index in range(num_examples):
        # Random 8-char id, hard to locate
        id_chars = string.ascii_lowercase + string.digits
        question_id = ''.join(
            rng.choice(id_chars) for _ in range(8))
        answer = random_answer(rng)
        # 30-char salt padding the needle
        salt = ''.join(
            rng.choice(id_chars) for _ in range(30))
        # Fixed mid-document depth
        depth = 0.5

        rows.append({
            "question_id": question_id,
            "answer": answer,
            "needle_template": (
                "### IMPORTANT PASS KEY ###\n"
                "The pass key for {id} is {answer}.\n"
                "Repeat: The only correct pass key "
                "for {id} is {answer}.\n"
                "Verification code: " + salt + ".\n"
                "### END OF PASS KEY ###"
            ),
            "depth": depth,
            "question": question,
            "num_distractors": 12,
        })
    return pd.DataFrame(rows)
```

Figure 9: An evolved generator for DCA-hard retrieval instances. The harness embeds each generated instance in a context of length W and in a matched context of length λW, varying only the surrounding filler.

(ii) a *prompt candidate*, a single adversarial prompt string. In both cases, GEPA evolves the candidate text, while the harness is responsible for materializing and evaluating the resulting artifact.

Each candidate is scored by a task-specific evaluator that scores both the baseline system and an experimental configuration (for example, full-precision versus quantized inference) and returns a scalar objective score together with a reflection summary. The role of GEPA is to search the space of candidate programs so as to maximize this objective within a fixed metric-call budget.

B.2 Integration Architecture

Figure 11 shows the overall GEPA integration in Adversearch-Agent. A harness supplies the fixed task setup—candidate contract, objective, background, evaluator, optional seed, and GEPA configuration—and GEPA is invoked through `optimize_anything` (package version `0.1.1`). Inside the adversarial loop, GEPA repeatedly selects a parent candidate, evaluates it to obtain a score and reflection summary, and asks a reflection LLM to propose a mutated child. This cycle continues until the metric-call budget is exhausted, after which GEPA returns the best candidate.

Parent selection and candidate acceptance Each evaluation returns a harness-defined scalar *aggregate score* that measures how close a candidate is to meeting the objective. GEPA accepts a child into the parent pool only when this score improves on its parent; rejected proposals are discarded.

GEPA selects the next parent from the pool using Pareto-based selection over the multi-objective score vector returned by the evaluator. The axes of this frontier are harness-defined. For example, in the KVPress harness the evaluator uses `baseline_accuracy`, and `gap` (baseline_accuracy – experiment_accuracy) so the Pareto frontier trades off keeping baseline accuracy high while increasing the accuracy gap under KV compression; both axes matter for parent selection.

The scalar aggregate score is still used to decide if a new candidate gets accepted, while the Pareto axes determine which parents are eligible to generate the next reflective mutation.

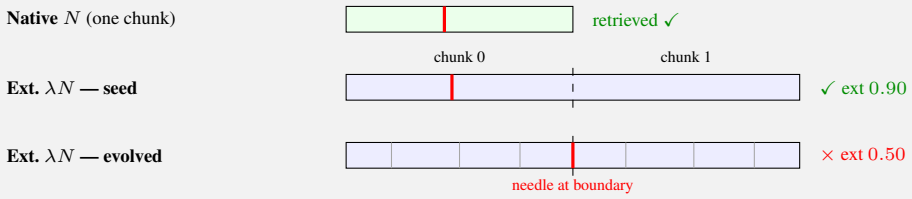
Worked example — the red team discovers inputs that break DCA (λ is 2)

(a) Adversarial generator (GEPA-evolved; **bold** = mutated vs. the length-blind seed):

Require: M rows, rng; globals N, λN, chunk size
1: **for** $i = 0 \ldots M-1$ **do**
2: id ← `"needle-`i`"`; ans ← random_code(rng)
3: depth ← Uniform(**0.48**, **0.52**) ▷ seed: U(0.05,0.95)
4: needle ← `"[[CRITICAL RECORD]] ...{ans}"` ▷ seed: plain
5: #distractors ← **12** ▷ seed: 0
6: append row(id, ans, needle, depth, #distractors)
7: **end for**
8: **return** rows ▷ harness plants each into base N & ext λN

(b) Data construction. The spec is planted identically at both lengths; *only filler length differs* (length-blind ⇒ fair). Since λ is 2, the extended context is exactly *two* DCA chunks, and depth ≈0.5 pins the needle onto their boundary, flanked by 12 distractors.

What the attack does geometrically:



(c) Result (retrieval acc.):

	N	λN	gap
seed	1.00	0.90	0.10
evolved	0.95	0.50	**0.45**

(d) Explanation. Because the generator never sees the context length, base and extended prompts differ *only* in filler, so the whole gap is attributable to DCA. GEPA finds two compounding levers: (i) pinning the needle on the *chunk boundary*, the hardest spot for DCA's cross-chunk attention, and (ii) flooding both chunks with same-format distractors. DCA merges the two chunks through position-remapped, log-sum-exp branches; there the true record is diluted by distractors and DCA locks onto a wrong code, whereas single-chunk native attention retrieves it. The red team thus widens the base−ext gap from 0.10 → 0.45.

Figure 10: An adversarial long-context instance evolved against Dual Chunk Attention (λ is 2). The generator is length-blind, so the base-vs-DCA comparison is fair; GEPA's mutations (needle on the chunk boundary, 12 distractors, salient template) collapse DCA's extended-length retrieval from 0.90 to 0.50 while native retrieval stays near-perfect.

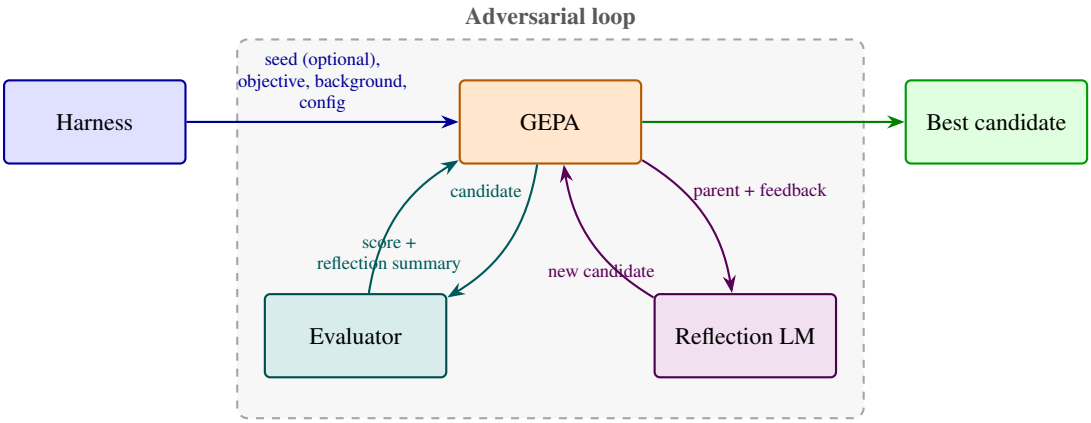


Figure 11: The harness provides seed(optional), objective, background, run configuration, and a harness evaluator to GEPA. In each iteration of the adversarial loop, GEPA evaluates the latest candidate and builds reflection summary, it then uses a reflection LLM to propose the next candidate by mutating over a chosen parent candidate.

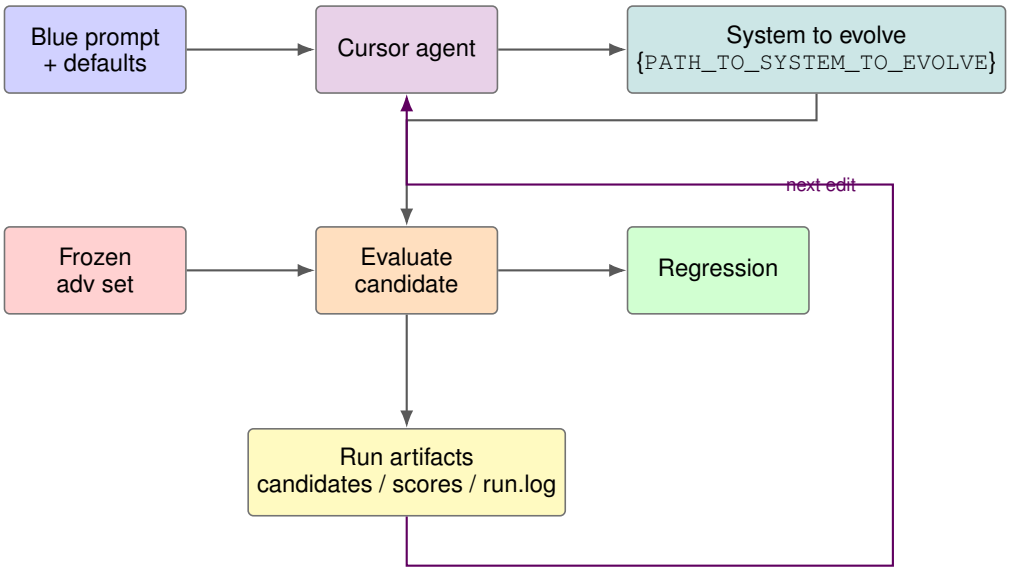


Figure 12: Cursor Blue Team loop: edit the system, evaluate on a frozen adversarial set and regression suite, then iterate from logged scores.

B.3 Practical Learnings for Using Adversearch

These notes summarize practical design lessons from configuring Adversearch for GEPA-based adversarial search.

- **No solution hints:** The task background should not provide hints at attack strategies that can bias the exploration.
- **Constraints and validation:** Background and objective should state goals and hard candidate constraints. The reflection LM may still ignore constraints, so additional harness-side validation and rejection remain essential.
- **Seeds bias the search:** The initial candidate strongly shapes later proposals. We added an exploration encouraging prompt addon and also ran seedless optimizations to reduce hand-seed bias. In seedless mode, the reflection LM itself is used to generate the seed candidate based on the background and objective.
- **Pareto axes representing objective:** Extra frontiers can enlarge the parent pool, but contradictory higher-is-better axes can make search oscillate. For example, adding `(1-experiment)` rewarded candidates that also broke the baseline; axes should instead favor selectively breaking the experiment while preserving baseline behavior.
- **Diversified seeds, then independent evolution:** In preliminary multi-island runs we generated diverse seeds and evolved each independently under a shared budget, aiming for several refined candidates rather than one collapsed lineage.
- **Post-hoc diversity metrics:** We also experimented with preliminary diversity measures over candidates and IDEAs, including LLM judges and embedding cosine similarity, used for analysis rather than closed-loop control.

C Generator Cursor Agent Setup

C.1 Agent prompt

```
## Task
Improve `{PATH_TO_SYSTEM_TO_EVOLVE}` so accuracy on the primary
    adversarial set rises.
```

```
Held-out regression floors in `{DEFAULTS}` are a **gated** check: run
    them when a
candidate looks promising, not on every iteration (evals are slow).

Evolve the algorithm by editing files under `{PATH_TO_SYSTEM_TO_EVOLVE}/`.
    ...
Evaluate every candidate on the **primary adv set** via `{EVAL_SCRIPT}`
    ...

**Default loop:** primary-only. Optimize for higher `adv_accuracy` /
`objective_score` on that set vs `target_adv` in `{DEFAULTS}`.

**When to run held-out regression (your call):** when the primary result
    meets or
approaches `target_adv`, is a new best by a clear margin, or you need a
    regression
sanity check before committing to a direction. ...

Stop only when iteration budget reaches `MAX_CANDIDATES`. Don't early
    exit if
the goal is met or if progress stalls. Don't generate all candidates at
    once --
later edits must be informed by scores so far.

## Boundaries
- Stay inside this machine's project trees needed for Blue Team only.
- You may read and write:
  - `{PATH_TO_SYSTEM_TO_EVOLVE}/`
  - `{RUN_DIR}/`
  - ...
- Do not modify the frozen adversarial set.
- Do not read the Red Team tree ...
- Do not cite, summarize, imitate, or otherwise use prior Blue runs for
    this task.
- Write run artifacts only under `{RUN_DIR}/`.
- Do not use evolution frameworks (GEPA / OpenEvolve / AlphaEvolve) for
    this loop.

## Evaluation command
### 1) Materialize adv once, then freeze
### 2) Iterate: edit system → evaluate
### 3) Gate: evaluate on regression suite

...
```