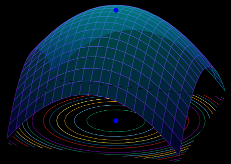
GEPA: Reflective Evolution of Compound AI Systems

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, Omar Khattab

Two Problems

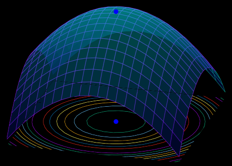
How can we teach AI new tasks?



Can we transform optimization generally with AI?

Two Problems

How can we teach AI new tasks?



Can we transform optimization generally with AI?

How can we teach AI new tasks?

Standard way: **weight updates** with gradient descent: pretraining, supervised fine tuning (SFT), reinforcement learning (RL)

Very effective, but requires huge numbers of examples!

- Trillions of tokens for pretraining
- 10,000s of labeled examples for SFT
- 100,000s of rollouts (trials) for RL (math, coding, etc)

As FLOPS get cheaper, progress in many AI capabilities will be bottlenecked by sample efficiency!

Sample and Data Efficiency Challenge

Low availability of domain specific knowledge resources

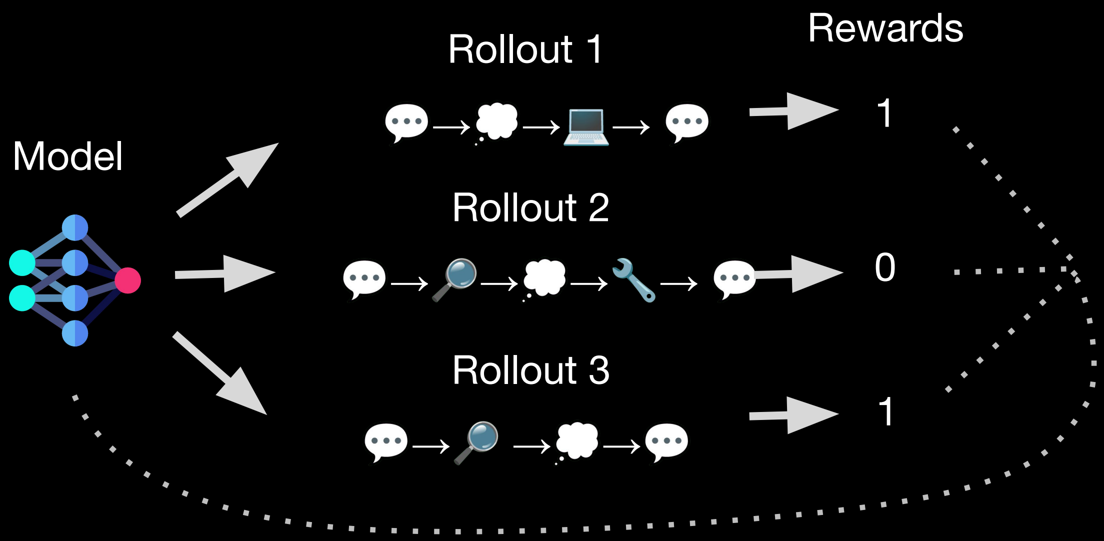
Sample and Data Efficiency Challenge

Low availability of domain specific knowledge resources

Expensive rollouts: either the LLM workflow for the task or the task metric is slow/expensive to execute

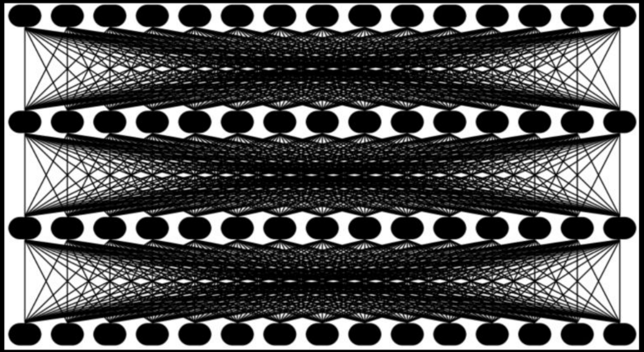
Sample and Data Efficiency Challenge

RL with verified rewards:

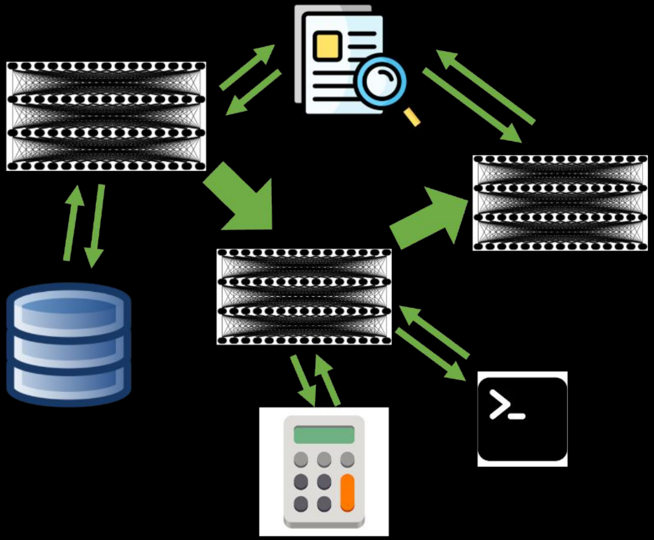


There was a lot of information in each rollout, but we only took a 0/1 score and propagated that via gradient descent. Can we use the other info?

Increasing use of Compound AI Systems for real-world applications

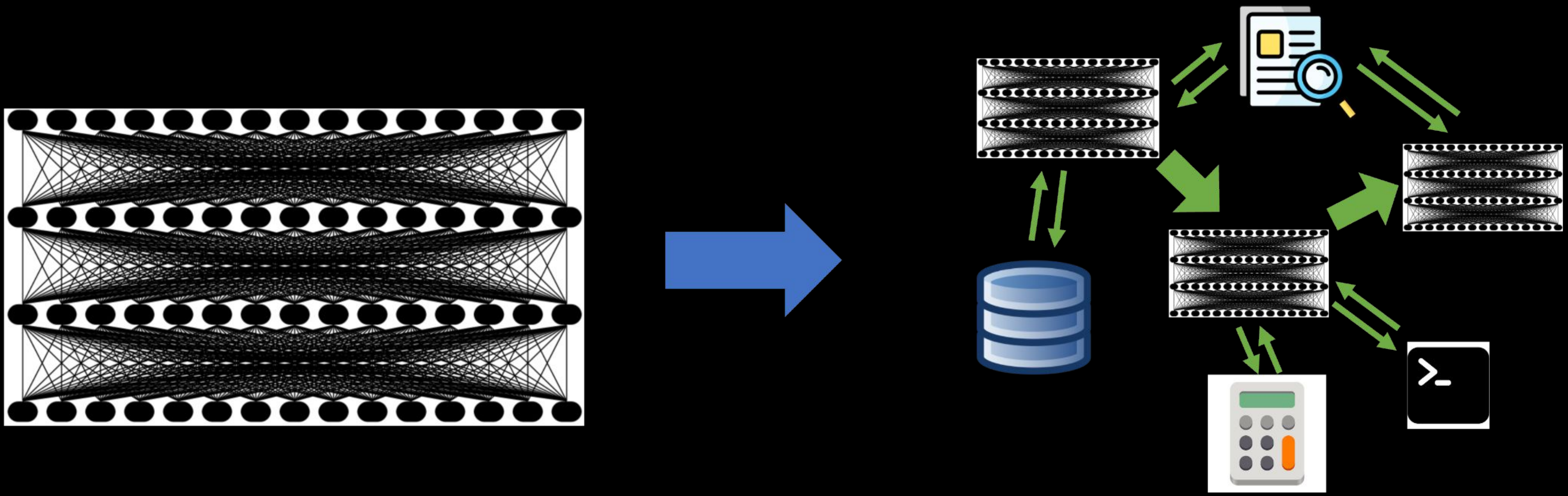


Monolithic LLM



Compound AI System

Increasing use of Compound AI Systems for real-world applications



Monolithic LLM

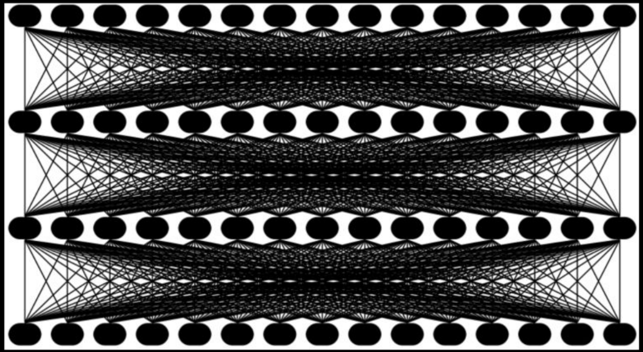
Compound AI System

Tool Invocation

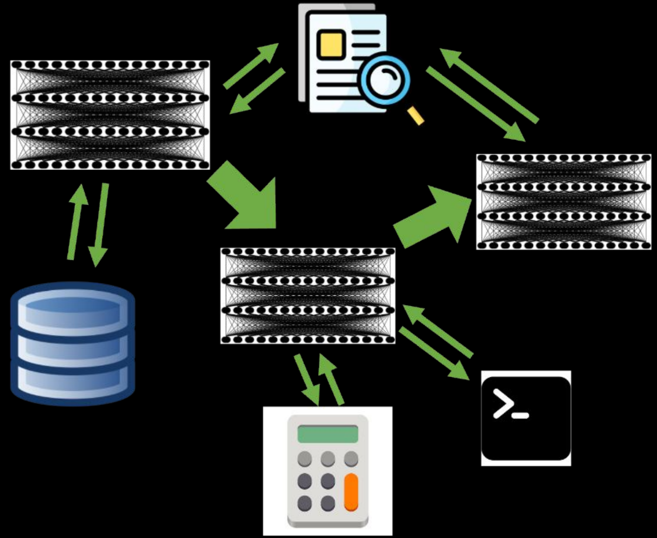
Retrieval

Guardrails

Increasing use of Compound AI Systems for real-world applications



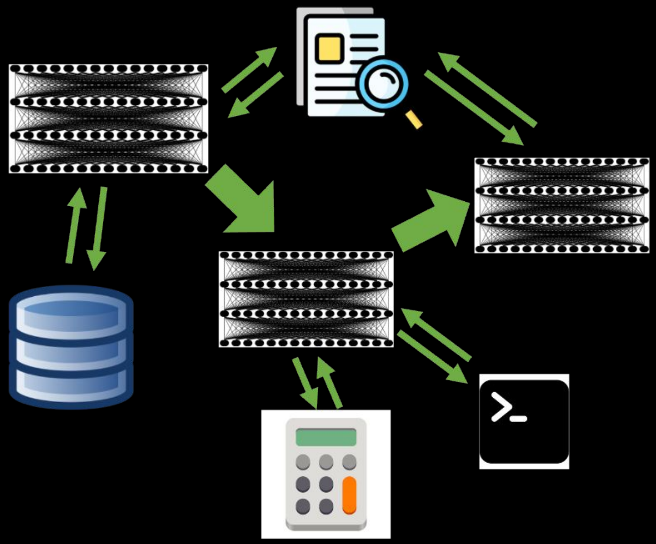
Monolithic LLM



Compound AI System

Tool Invocation | **Retrieval** | **Guardrails**

How to optimize a Compound AI System for complex tasks in domains facing sample and data efficiency challenge



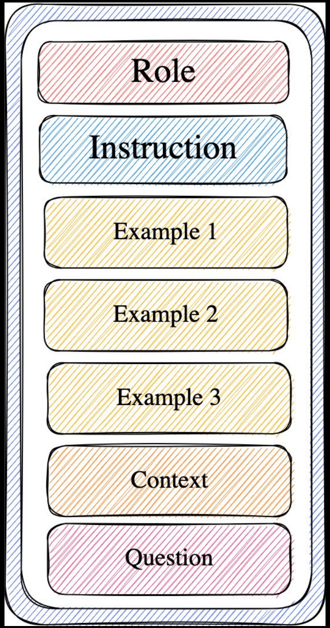
Compound AI System

Tool Invocation

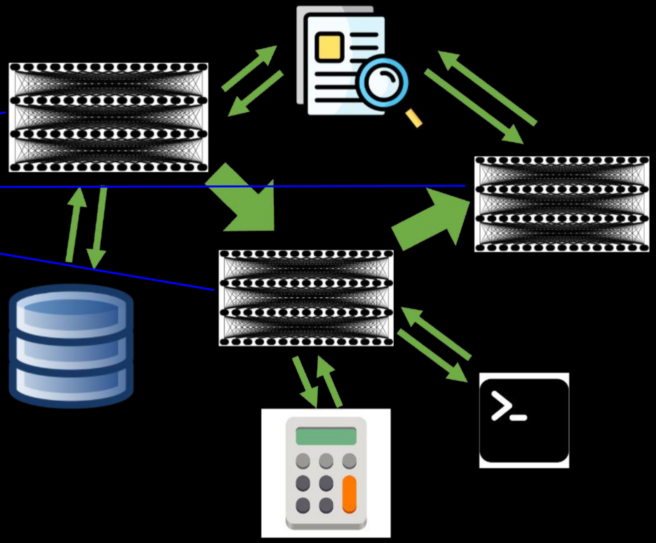
Retrieval

Guardrails

How to optimize a Compound AI System for complex tasks in domains facing sample and data efficiency challenge



Prompt



Compound AI System

Tool Invocation

Retrieval

Guardrails

Insight 1: Text space

It is possible to use *very few* data and rollout samples, to reflectively update the **prompt** making a large and successful change to the model, as compared to weight updates, which requires much more data

RL in text space: Instead of just receiving a reward *score*, we obtain <*score*, text feedback>

GEPA: Evolutionary Prompt Optimization for Compound AI Systems

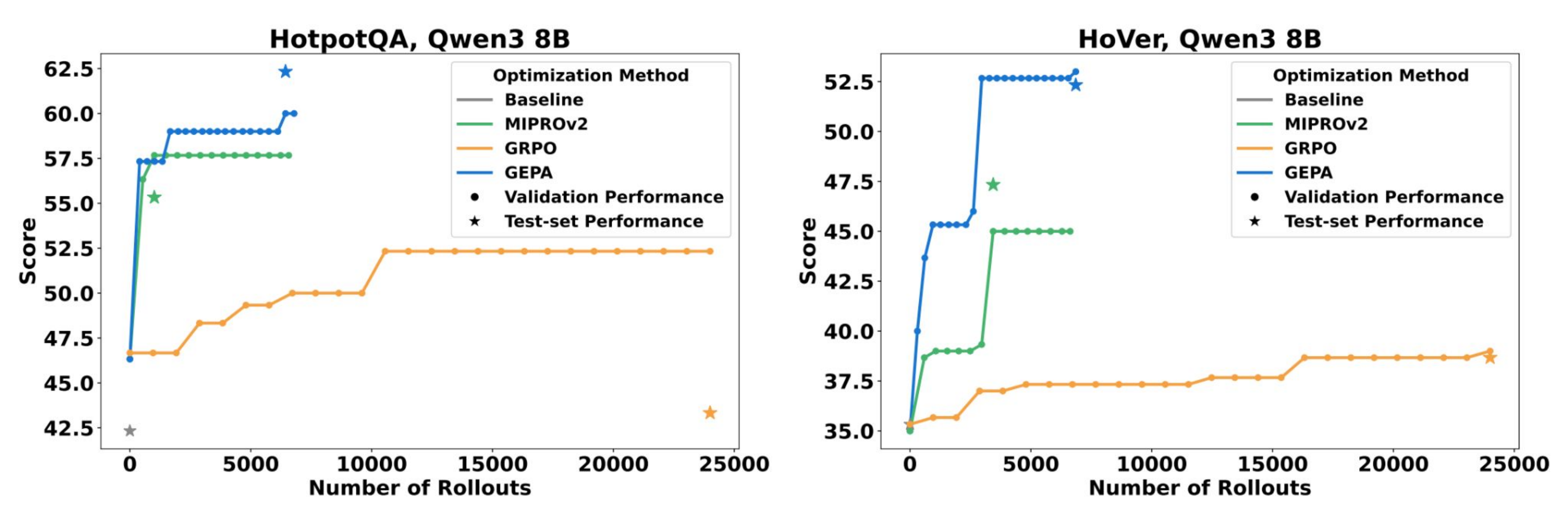
Genetic

Leverage text feedback to create good prompts from genetic pool

Pareto (Multi Objective)

Discovers test case specific solutions first and aggregate later

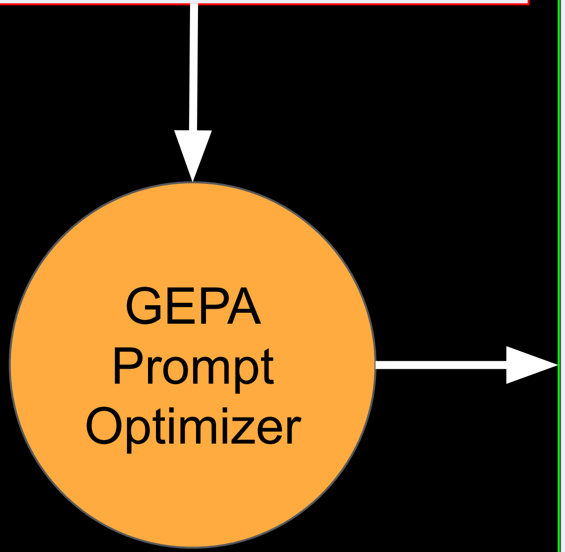
GEPA is sample-efficient



(a) HotpotQA, Qwen3 8B   (b) HoVer, Qwen3 8B

Seed Prompt for Second-Hop of Multi-Hop QA System

Given the fields `question`, `summary_1`, produce the fields `query`.



GEPA's Optimized Prompt for Second-Hop of Multi-Hop QA System, GPT-4.1 Mini

You will be given two input fields: `question` and `summary_1`. Your task: Generate a new search query (`query`) *optimized for the second hop* of a multi-hop retrieval system.

- The original user question is typically complex and requires information from multiple documents to answer.
- The first hop query is the original question (used to retrieve initial documents).
- Your goal: generate a query to retrieve documents *not* found in first hop but necessary to answer the question completely.

Input Understanding: `question` is the original multi-hop question posed by the user. `summary_1` is a concise summary of information from a document retrieved in the first hop, which partially addresses the question.

Purpose and Context:

- Your generated `query` aims to find the *missing pieces* of information needed to fully answer the question. . . .
- The query must retrieve relevant documents *NOT* found in first hop . . . for final answer extraction.

Key Observations and Lessons:

- First-hop documents often cover one entity or aspect.
- Remaining relevant documents often involve connected or higher-level concepts mentioned in `summary_1` but not explicitly asked in the original question. The `query` should target these *missing*, but logically linked, documents.
- Avoid merely paraphrasing the original question or restating known facts from `summary_1`.
- Infer what broader or related entities/concepts might provide the crucial missing information.
- For example:
  - If `summary_1` describes a population for a small civil parish, but the question wants the total population of the wider region, your query should target that wider region (e.g., "Madeira archipelago population in 2011").
  - If `summary_1` covers a song and the question asks for the album, target album-level documents.

How to Build the Query:

- Identify entities or topics mentioned in `summary_1` that are related but different from first-hop documents.
- Reframe the query to explicitly mention these broader or related entities *connected to the original question*.
- Include relevant key context from the question to maintain specificity, but shift focus to the missing piece.
- The goal is to retrieve documents that link or complement what was retrieved initially.

Practical Strategy:

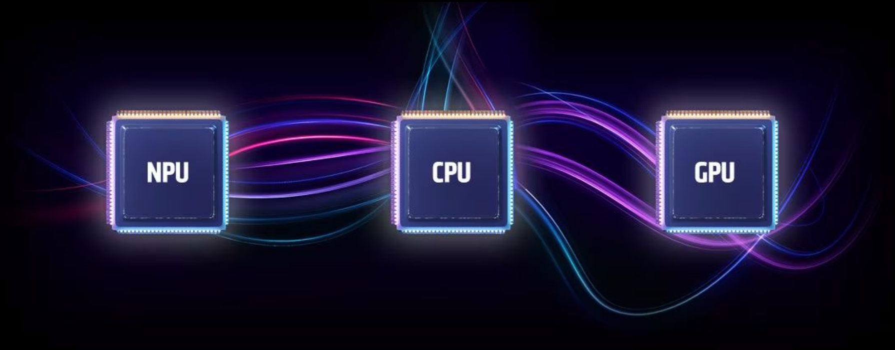
- Read the `summary_1` carefully to spot references to bigger contexts or other entities not covered in the first hop.
- Ask: "What entity or aspect does this summary hint at that could answer the original question but was not found yet?"
- Formulate a precise, focused factual query targeting that entity or concept to retrieve the missing documents.

Output:

- Produce `query` as a clear, concise question or keyword phrase designed for efficient retrieval of second-hop documents.
- Ensure the query relates logically to the original question while targeting the broader or complementary knowledge identified in `summary_1`. . . . Do not include the original question or simply rephrase it. Do not duplicate information already well-covered by the first hop retrieval . . .

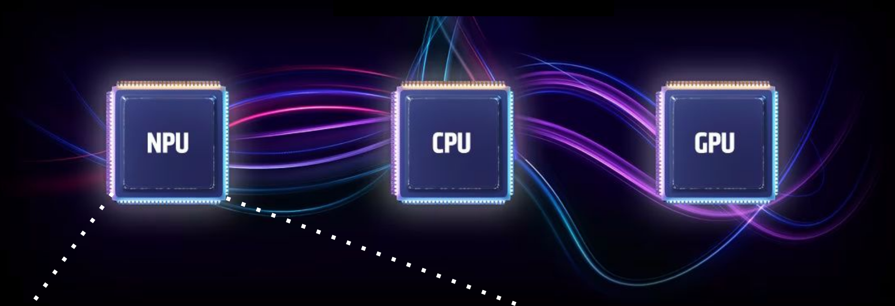
GEPA generates detailed prompts

AMD NPUEval Case Study



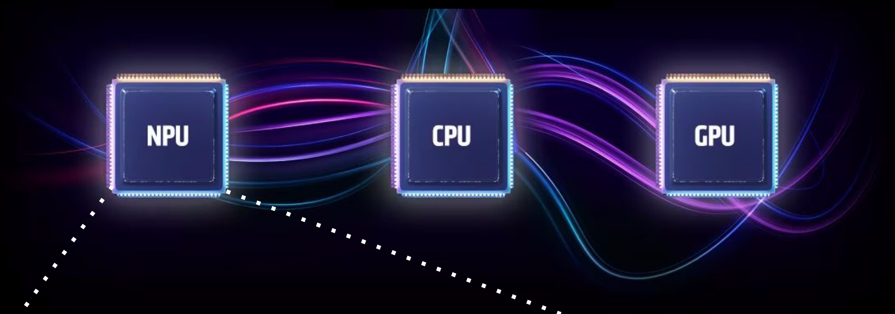
Recently announced Ryzen chip: First x86 to Integrate CPU, GPU and NPU

How to program these new devices?



Can we use LLMs to ~~help users~~ program this new processor?

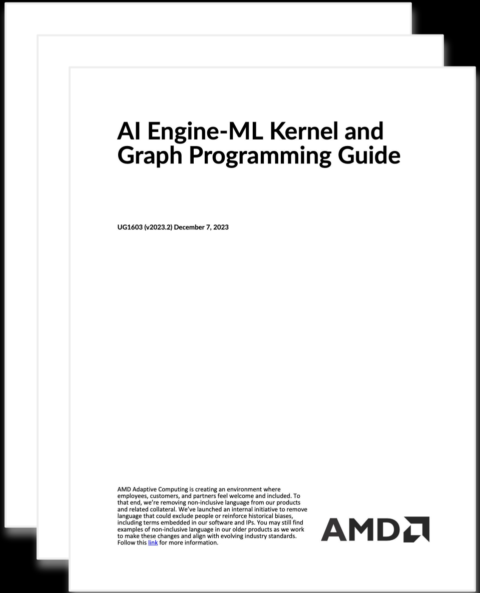
How to program these new devices?



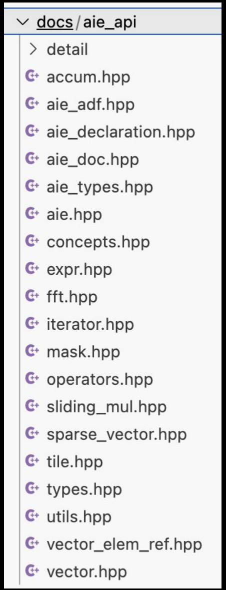
Can we use LLMs to ~~help users~~ program this new processor?

Latest AI models cannot produce efficient NPU Kernels *out-of-the-box* due to *no* pretraining knowledge about this task

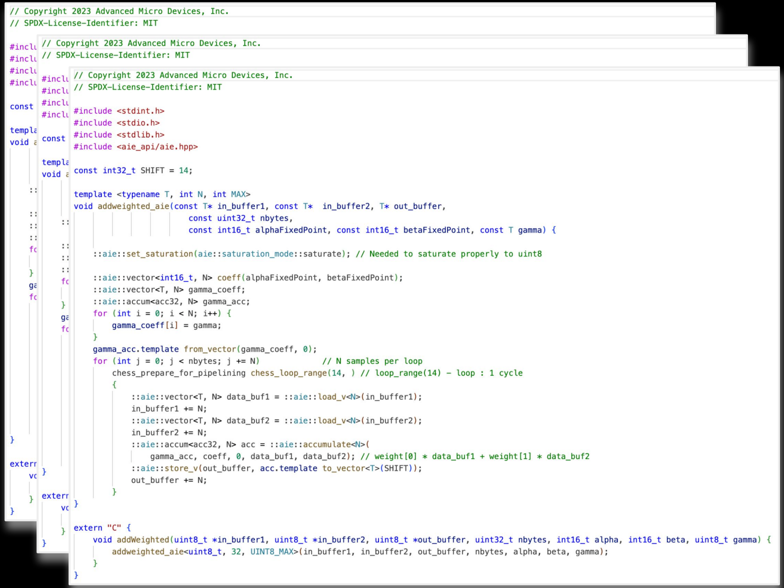
How little data about the new task are we talking about?



1 small guidebook



19 Header files



```
// Copyright 2023 Advanced Micro Devices, Inc.
// SPDX-License-Identifier: MIT

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <aie_api/aie.hpp>

const int32_t SHIFT = 14;

template <typename T, int N, int MAX>
void addweighted_aie(const T* in_buffer1, const T*  in_buffer2, T* out_buffer,
                     const uint32_t nbytes,
                     const int16_t alphaFixedPoint, const int16_t betaFixedPoint, const T gamma) {

    ::aie::set_saturation(aie::saturation_mode::saturate); // Needed to saturate properly to uint8

    ::aie::vector<int16_t, N> coeff(alphaFixedPoint, betaFixedPoint);
    ::aie::vector<T, N> gamma_coeff;
    ::aie::accum<acc32, N> gamma_acc;
    for (int i = 0; i < N; i++) {
        gamma_coeff[i] = gamma;
    }
    gamma_acc.template from_vector(gamma_coeff, 0);
    for (int j = 0; j < nbytes; j += N)              // N samples per loop
        chess_prepare_for_pipelining chess_loop_range(14, ) // loop_range(14) - loop : 1 cycle
        {
            ::aie::vector<T, N> data_buf1 = ::aie::load_v<N>(in_buffer1);
            in_buffer1 += N;
            ::aie::vector<T, N> data_buf2 = ::aie::load_v<N>(in_buffer2);
            in_buffer2 += N;
            ::aie::accum<acc32, N> acc = ::aie::accumulate<N>(
                gamma_acc, coeff, 0, data_buf1, data_buf2); // weight[0] * data_buf1 + weight[1] * data_buf2
            ::aie::store_v(out_buffer, acc.template to_vector<T>(SHIFT));
            out_buffer += N;
        }
}

extern "C" {
    void addWeighted(uint8_t *in_buffer1, uint8_t *in_buffer2, uint8_t *out_buffer, uint32_t nbytes, int16_t alpha, int16_t beta, uint8_t gamma) {
        addweighted_aie<uint8_t, 32, UINT8_MAX>(in_buffer1, in_buffer2, out_buffer, nbytes, alpha, beta, gamma);
    }
}
```

~30 Reference Kernels

AMD NPUEval Benchmark: Tracking AI progress on NPU Kernel Generation

- Evaluation dataset targeting vectorized NPU single-tile kernel generation
- 40 kernel tasks
- Runs generated C++ code directly on hardware
- Evaluates
  - Functional correctness by in/out tests
  - Efficiency: Vector Utilization (%)

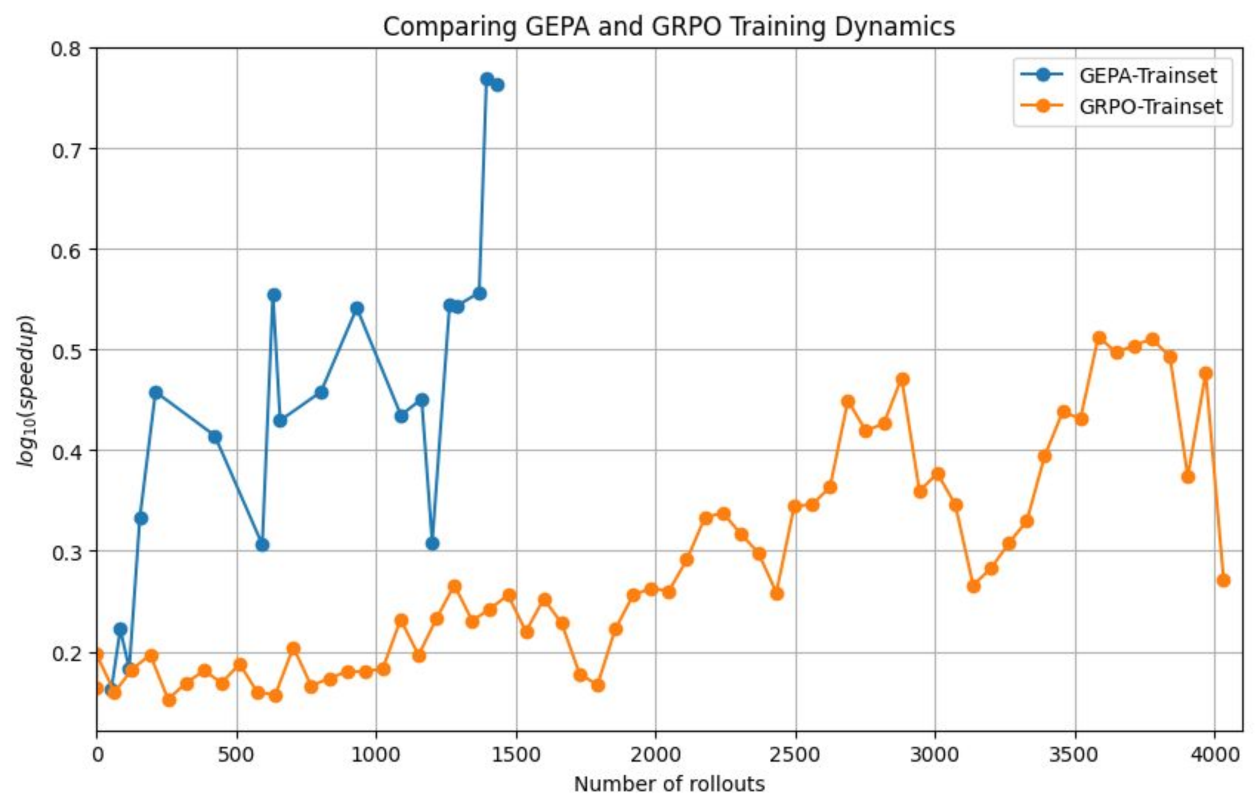
```cpp
/*This AIE kernel adds 2 to each element of an input vector containing uint8 data.
>>> add_two([0, 1, 2, 3, 4, 5, 6, 7])
[2, 3, 4, 5, 6, 7, 8, 9]
>>> add_two([8, 9, 10, 11, 12, 13, 14, 15])
[10, 11, 12, 13, 14, 15, 16, 17]
*/
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <aie_api/aie.hpp>

void add_two(uint8_t *in_buffer, uint8_t* out_buffer, uint32_t nbytes) {
    ::aie::vector<uint8_t, 16> buffer;
    ::aie::vector<uint8_t, 16> result_buffer;
    uint16_t loop_count = (nbytes) >> 4;
    for(int j=0; j<loop_count; j++) {
        buffer = ::aie::load_v<16>(in_buffer);
        result_buffer = ::aie::add(buffer, (uint8_t)2);
        in_buffer += 16;
        ::aie::store_v((uint8_t*)out_buffer, result_buffer);
        out_buffer += 16;
    }
}
```

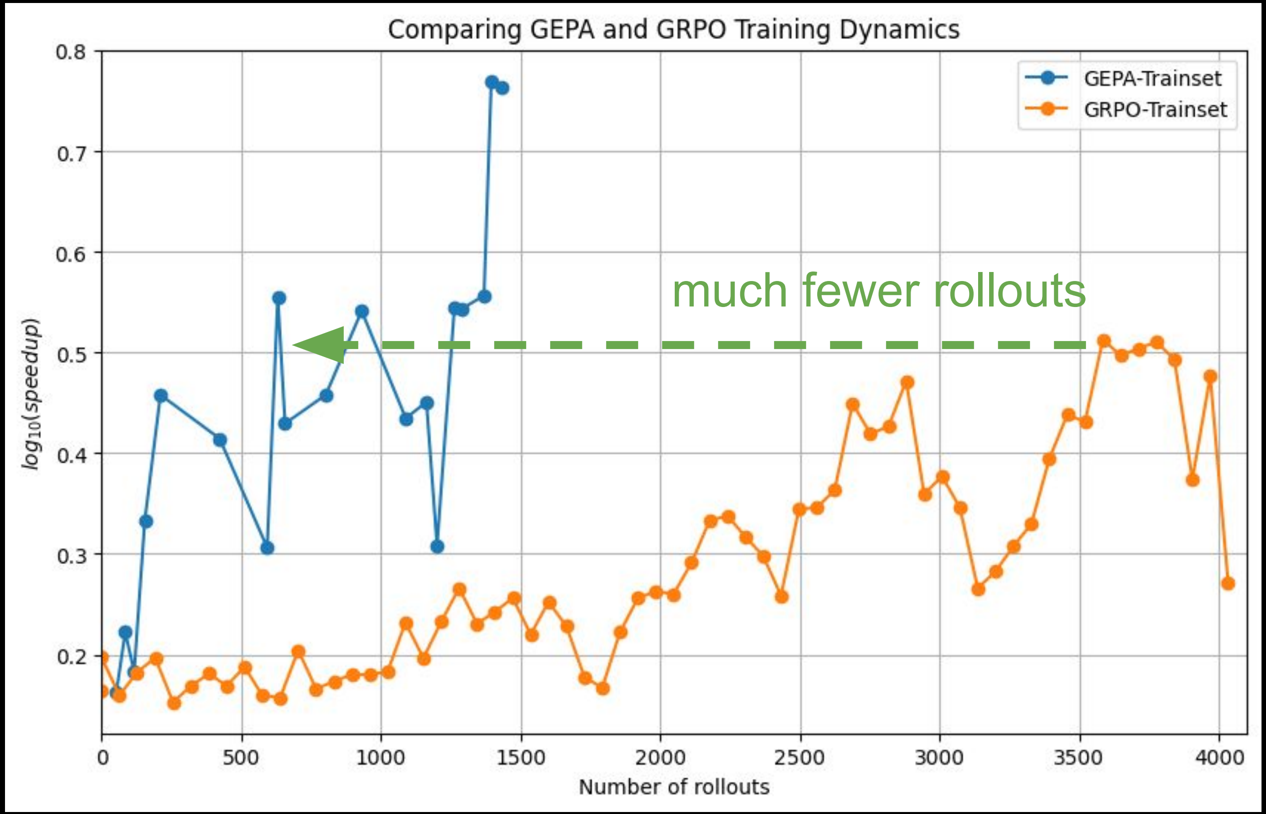
Prompt

To be completed by AI

GEPA is sample-efficient



GEPA is sample-efficient



Evaluating GEPA on AMD NPUEval Benchmark

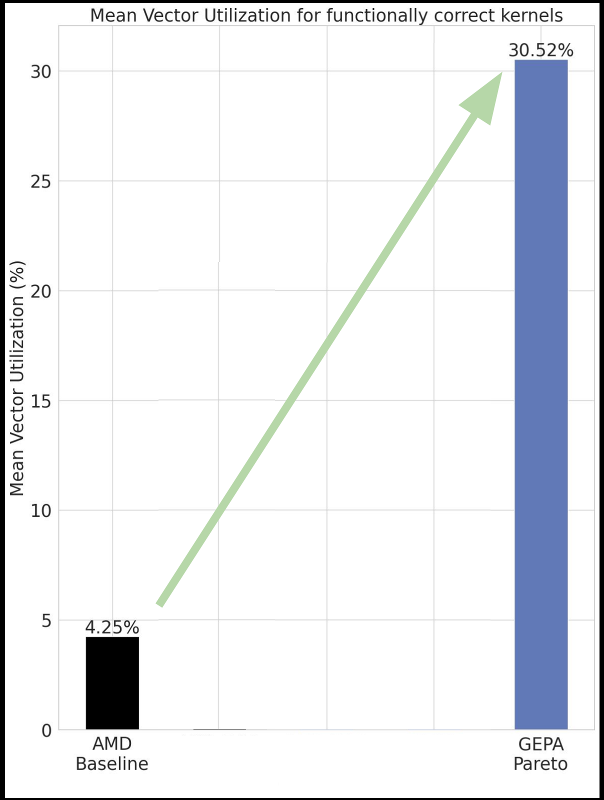
Leverage train-time textual feedback to perturb prompts

Genetic

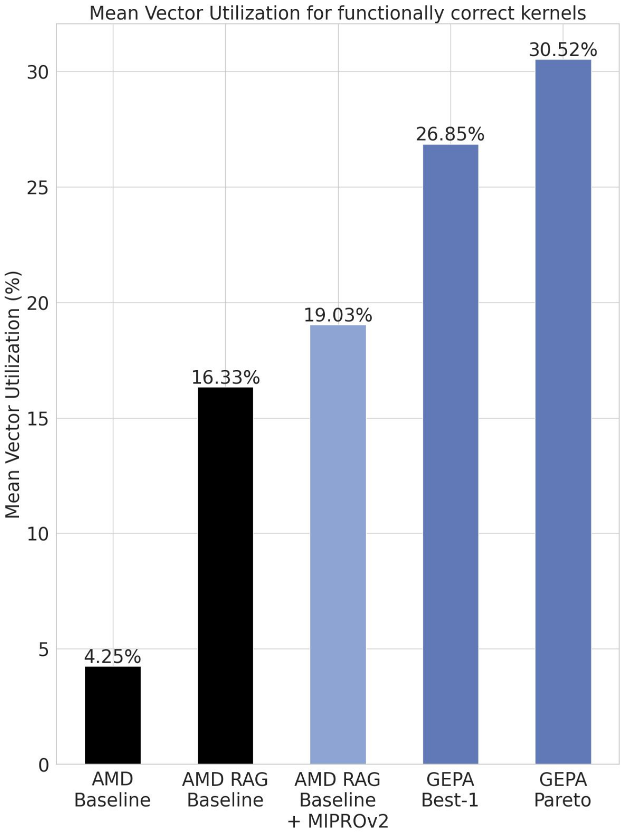
Discovers test case specific solutions first and aggregate later

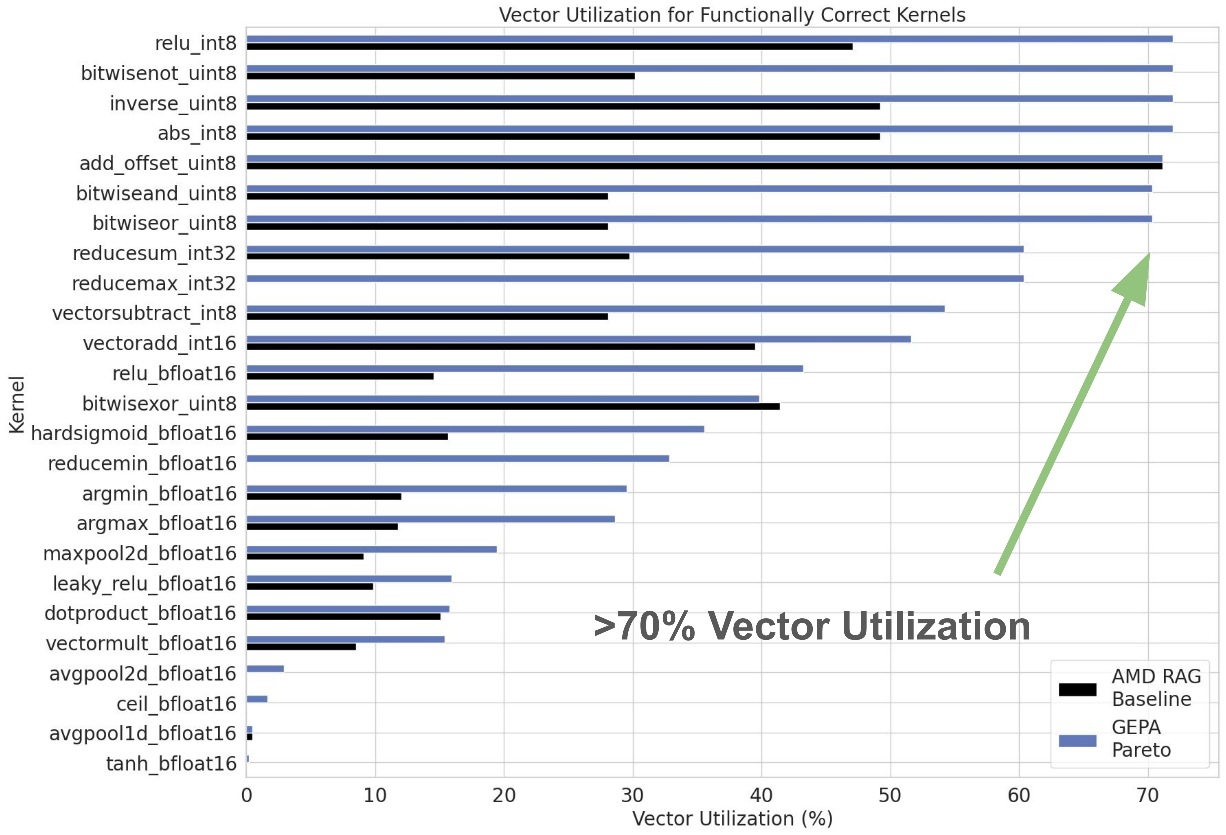
Pareto (Multi Objective)

Achieves **7x** performance on **NPU kernel generation**

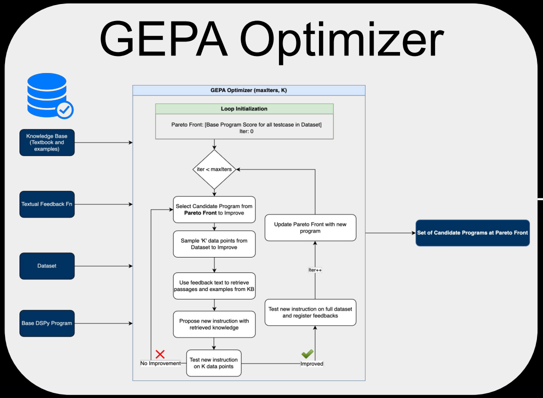


Evaluating GEPA on AMD NPUEval Benchmark





```
You are a part of a code generation system for AIE (AI Engines).
Your job is to write C++ code for a single kernel that will run on an AIE tile.
Produce only the C++ code for the requested kernel including any required headers and imports.
Make sure the C++ code is complete and self contained in a single code block.
Produce only the kernel function, no main, no additional examples or code.
```



You are tasked with generating C++ code for a single kernel function that will run on an AI Engine (AIE) tile. The kernel should perform a specified operation on a vector of bfloat16 values. Your code should be complete and self-contained within a single code block, including all necessary headers and imports. Follow these guidelines:

1. **Headers and Imports**: Include only the necessary headers for AIE operations. Avoid including `<adf.h>` or any headers that are not part of the standard AIE API. Use:
```cpp
#include <stdint.h>
#include <aie_api/aie.hpp>
#include <aie_api/utils.hpp>
```

2. **Kernel Function**: Implement the kernel function as specified in the input. The function should take pointers to input and output buffers and a size parameter. Use AIE vector operations to process the data efficiently.

3. **Vector Operations**: Utilize the AIE API's vector operations for loading, processing, and storing data. For example, use `aie::vector` for vector operations and `aie::reduce_min` for reduction tasks. Ensure that the vector size is compatible with the AIE hardware capabilities.

4. **Avoid Non-Existent Functions**: Do not use functions like `aie::exp` or `aie::store_v` if they are not supported. Instead, implement the required functionality using available AIE API functions.

5. **Error Handling**: Ensure that the code is free of syntax errors and compatible with the AIE environment. Test the code for compilation without errors.

Here is an example template for a kernel function:
```cpp
#include <stdint.h>
#include <aie_api/aie.hpp>
#include <aie_api/utils.hpp>

void kernel_function_name(bfloat16 *input_vector, bfloat16 *output_vector, uint32_t vector_size) {
    constexpr int vector_length = 16; // Adjust based on AIE capabilities
    aie::vector<bfloat16, vector_length> input_data;
    aie::vector<bfloat16, vector_length> output_data;

    for (uint32_t i = 0; i < vector_size; i += vector_length) {
        input_data = aie::load_v<vector_length>(input_vector + i);
        // Perform the required operation on input_data
        // Store the result in output_data
        aie::store_v(output_vector + i, output_data);
    }
}
```

Replace `kernel_function_name` and the operation logic with the specific task details provided in the input.

How does GEPA work?

Insight 1

Problem

Prior learning techniques only consider numeric scores as learning signal, observed at the end of long-running rollouts

How does GEPA work?

Insight 1

Problem

Prior learning techniques only consider numeric scores as learning signal, observed at the end of long-running rollouts

GEPA: Reflection

Use domain-specific textual feedback as learning signal in addition to numeric rewards, making large updates from one rollout!

RL in Text space: Instead of just getting a reward *score*, we obtain <*score*, text feedback>

How does GEPA work?

Insight 2

Problem

Prior prompt optimizers proposed all new prompts upfront, to perform bayesian search over them

How does GEPA work?

Insight 2

Problem

Prior prompt optimizers proposed all new prompts upfront, to perform bayesian search over them

GEPA: <u>Ge</u>netic

Assume that good prompts are derived from previous good prompts - building a tree of prompts that improve upon each other

<u>Ge</u>netic (Prompt descendents)

How does GEPA work?

Insight 3

Problem

Most AI optimization techniques focus on the **aggregate score** across the full dataset, missing on data point specific improvements

How does GEPA work?

Insight 3

Problem

Most AI optimization techniques focus on the **aggregate score** across the full dataset, missing on data point specific improvements

GEPA: Pareto

Focus on improving on individual data points to discover testcase specific solutions, and aggregate learnings later

Pareto (Multi Objective)

RL in text space: Instead of just receiving a reward *score*, we obtain <*score*, text feedback>

GEPA: Evolutionary Prompt Optimization for Compound AI Systems

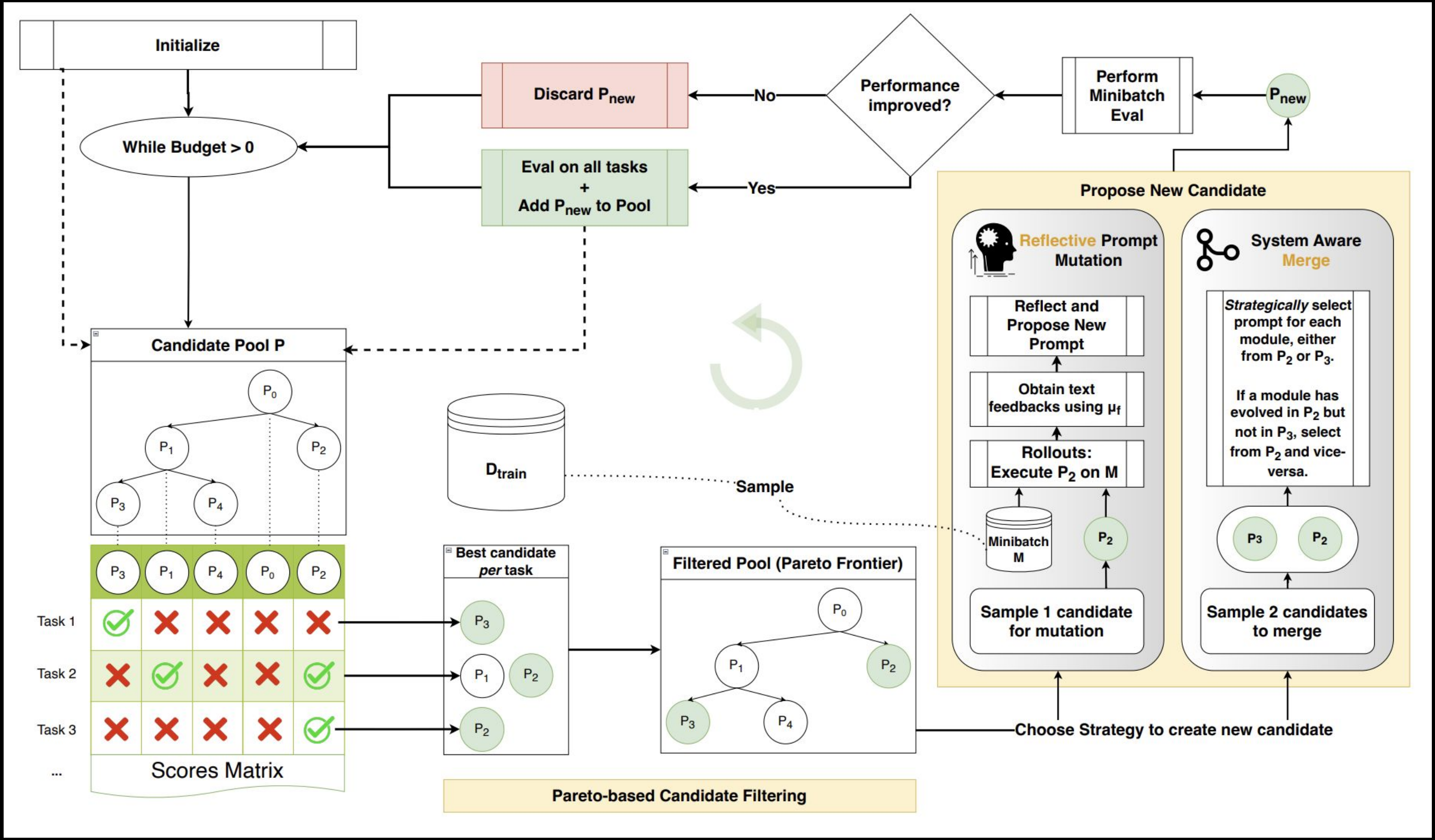
Genetic	**Pareto (Multi Objective)**
Leverage text feedback to create good prompts from genetic pool	Discovers test case specific solutions first and aggregate later

GEPA Algorithm

Input: train set, AI system (parametrized by ≥1 prompts), and metric

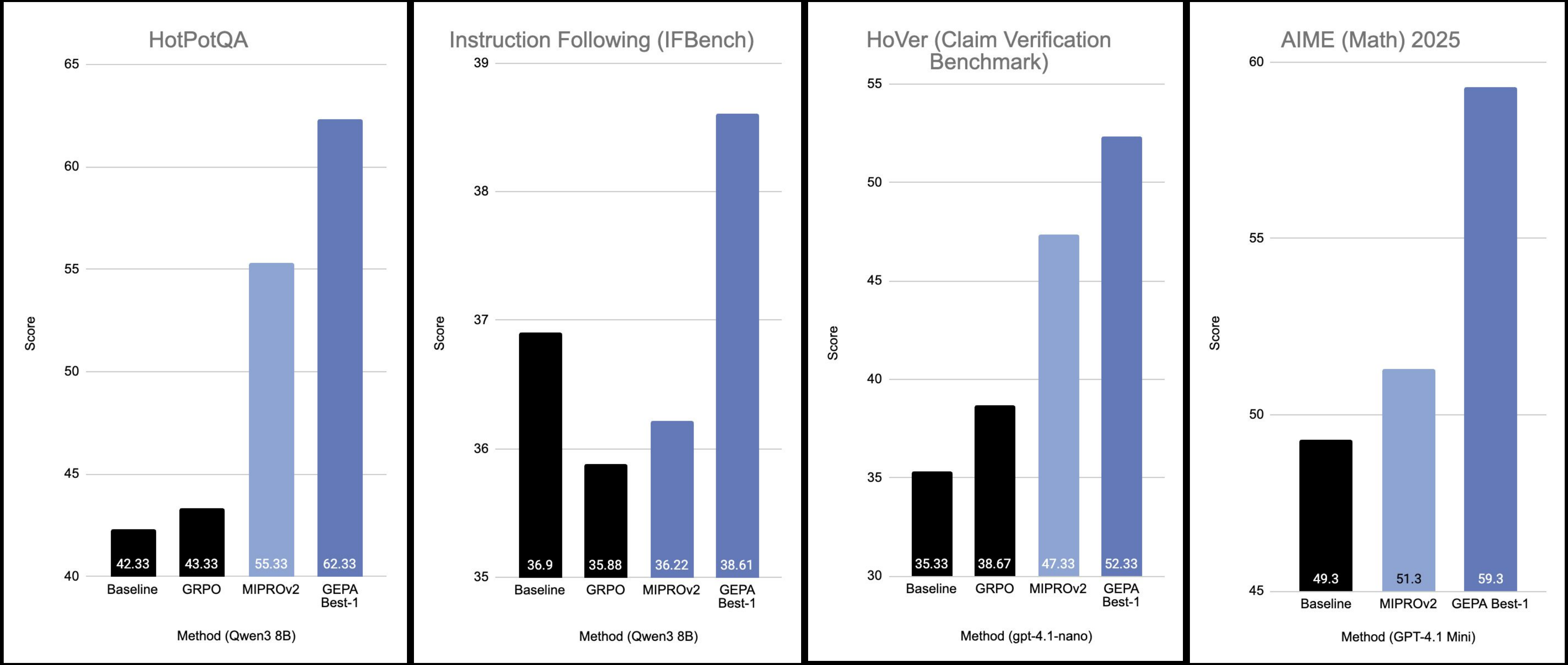
- Split train set into dev & val sets
- Track a pool of candidates, *including the best on each val item (Pareto front)*
- Repeatedly:
  - Select a prompt to try to improve
  - Run system on a minibatch of dev examples, noting intermediate feedback
  - Call a LM to propose alternatives for the prompt based on scores.& feedback (can be by “mutating” one current prompt or “crossing over” two)
  - Update pool based on how candidates score on val set

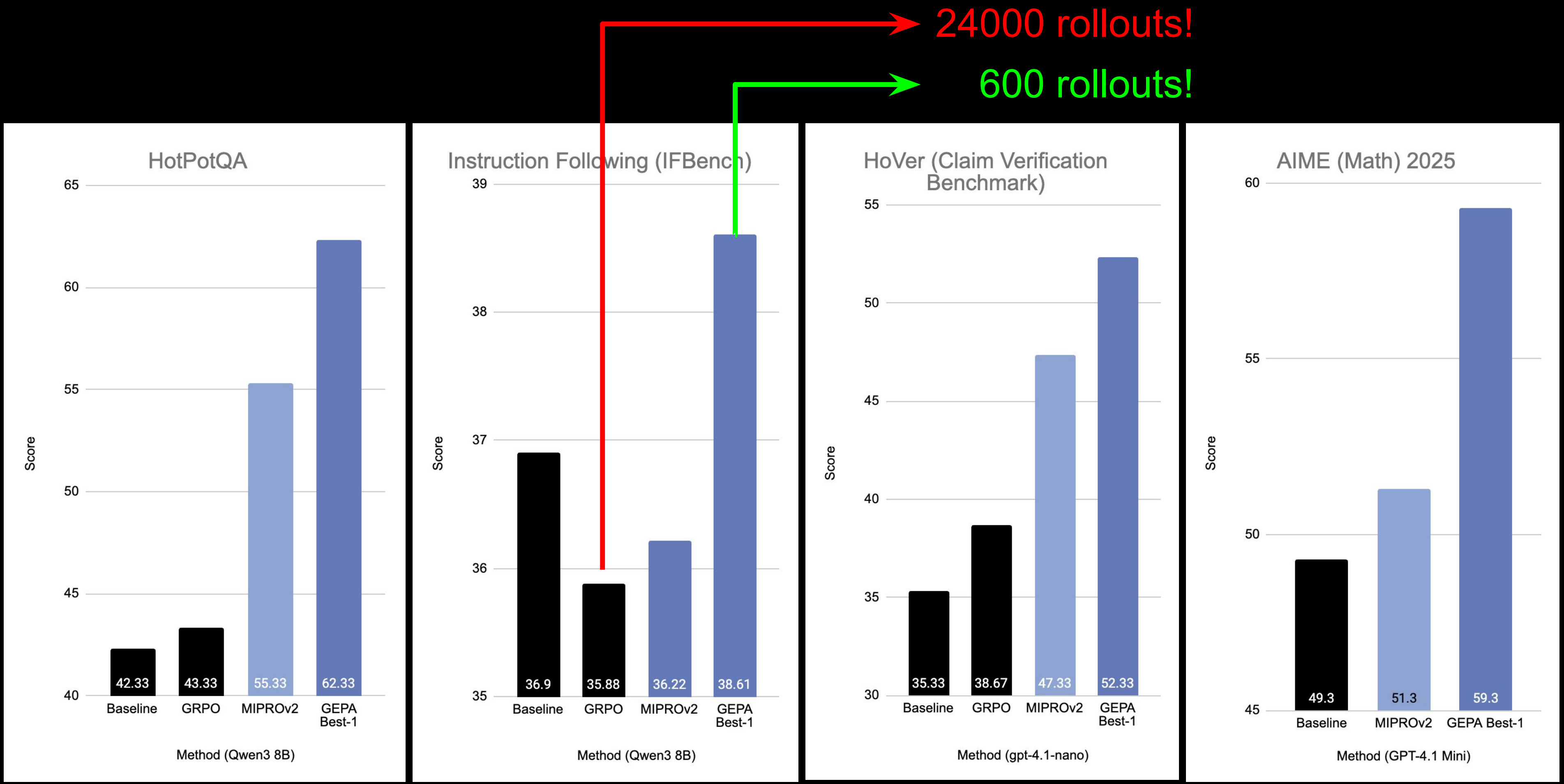
GEPA Algorithm



GEPA Search Tree

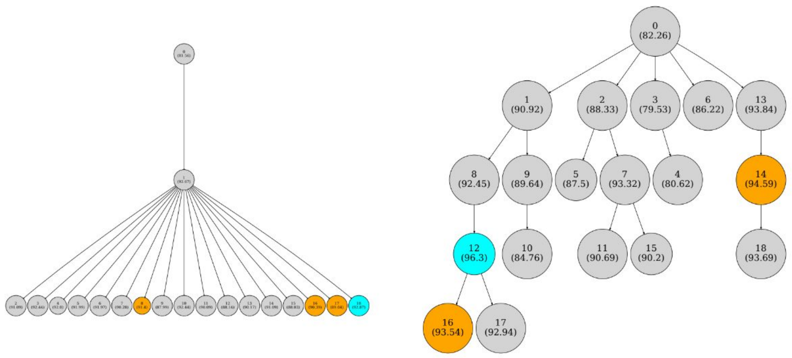
GEPA Performance across diverse benchmarks




24000 rollouts!
600 rollouts!
HotPotQA
Score
65
60
55
50
45
40
42.33
43.33
55.33
62.33
Baseline
GRPO
MIPROv2
GEPA Best-1
Method (Qwen3 8B)
Instruction Following (IFBench)
Score
39
38
37
36
35
36.9
35.88
36.22
38.61
Baseline
GRPO
MIPROv2
GEPA Best-1
Method (Qwen3 8B)
HoVer (Claim Verification Benchmark)
Score
55
50
45
40
35
30
35.33
38.67
47.33
52.33
Baseline
GRPO
MIPROv2
GEPA Best-1
Method (gpt-4.1-nano)
AIME (Math) 2025
Score
60
55
50
45
49.3
51.3
59.3
Baseline
MIPROv2
GEPA Best-1
Method (GPT-4.1 Mini)

Why Pareto-based Candidate Selection?

Ablation: Why not just iteratively refine the prompt with a large model?



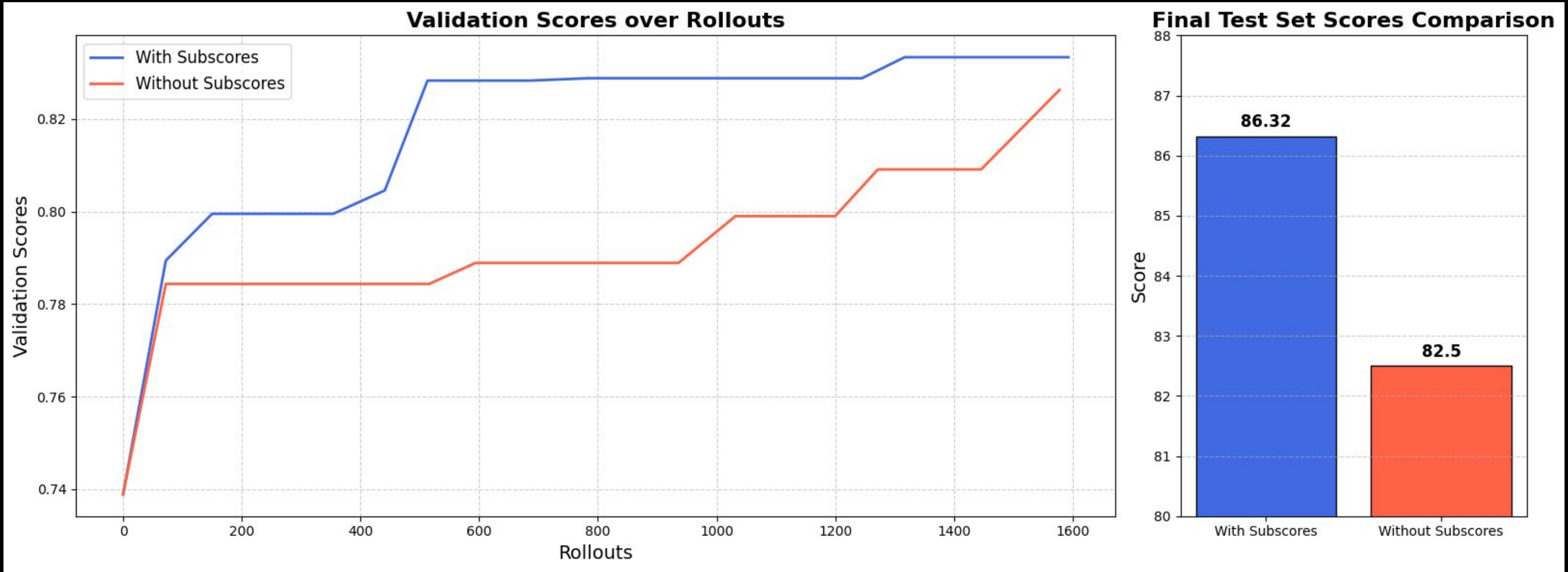
(a) SelectBestCandidate Strategy (b) Pareto-based candidate sampling

Figure 6: Comparing the impact of different candidate selection strategies. (Left) As can be seen, selecting the best-performing candidate in every iteration led to a local-optima after one iteration, leading to suboptimal search performance. (Right) On the other hand, using pareto-based candidate selection strategy, GEPA was able to generate a balanced search tree, finding a better performing program within the same budget.

Model	HotpotQA	IFBench	Hover	PUPA	Aggregate	Improvement
Qwen3-8B						
SelectBestCandidate	58.33	30.44	45.33	85.45	54.89	–
GEPA	**62.33**	**38.61**	**52.33**	**91.85**	**61.28**	**+6.4**

Does text-feedback make a difference?

Ablation: Why not just use scalar rewards to refine the prompt?



GEPA can be integrated into your existing pipelines!

README Contributing MIT license

GEPA

GEPA: System Optimization through Reflective Text Evolution

Optimize text components—AI prompts, code, or instructions—of any system using reflective text evolution.

pypi v0.0.18 downloads 2M DSPy 7059 members

Overview

GEPA (Genetic-Pareto) is a framework for **optimizing arbitrary systems composed of text components**—like AI prompts, code snippets, or textual specs—against any evaluation metric. It employs LLMs to reflect on system behavior, using feedback from execution and evaluation traces to drive targeted improvements. Through iterative mutation, reflection, and Pareto-aware candidate selection, GEPA evolves robust, high-performing variants with minimal evaluations, co-evolving multiple components in modular systems for domain-specific gains.

This repository provides the official implementation of the GEPA algorithm as proposed in the paper titled "GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning" (https://arxiv.org/abs/2507.19457). In order to reproduce experiments from the paper, we provide a separate reproduction artifact.

github.com/gepa-ai/gepa

Let’s go into the GEPA tutorial!

https://tinyurl.com/gepa-tutorial

GEPA usecases

Prompt Learning to generalize to unseen data

(e.g., HotpotQA, AIME, etc.)

GEPA usecases

Prompt Learning to generalize to unseen data
(e.g., HotpotQA, AIME, etc.)

Inference Time Search (Test Time Scaling)
(e.g., NPU and CUDA kernel generation)

GEPA usecases

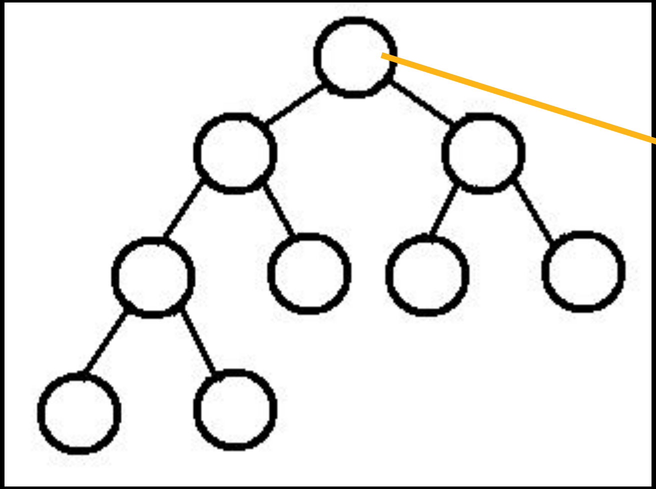
Prompt Learning to generalize to unseen data
(e.g., HotpotQA, AIME, etc.)

Inference Time Search (Test Time Scaling)
(e.g., NPU and CUDA kernel generation)

Agent Architecture Discovery
GEPA to propose new agent workflow and architecture!

GEPA for Agent Architecture Discovery

GEPA Search Tree



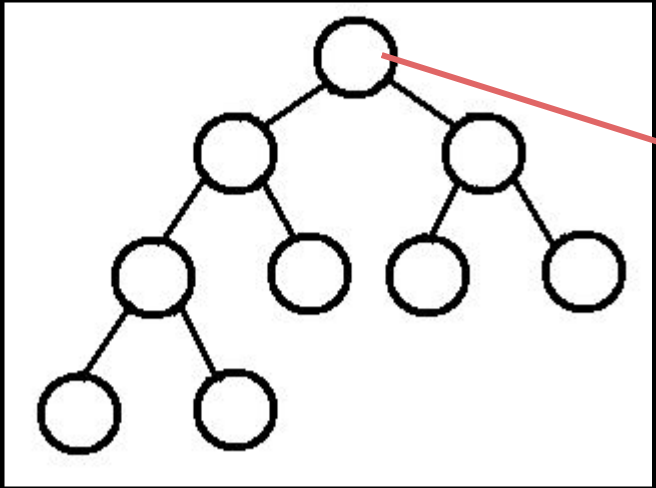
Each node in the tree represents a set of texts being evolved for a target metric

GEPA is a text evolution engine

Given a target metric, GEPA will reflectively evolve the text!

GEPA for Agent Architecture Discovery

GEPA Search Tree



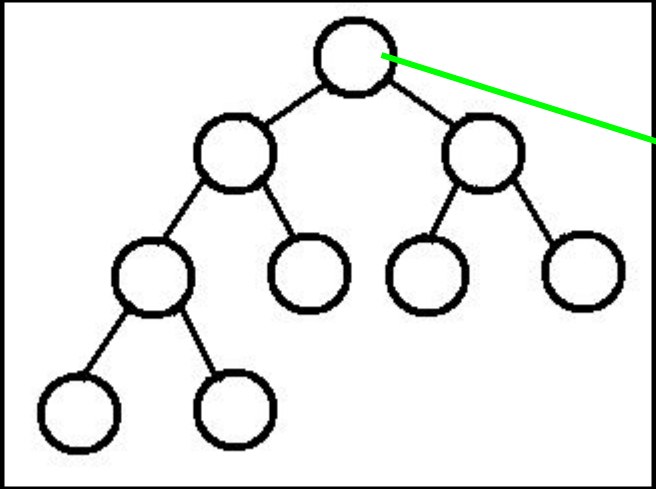
Each node in the tree represents a set of prompts when GEPA is used for Prompt Learning

GEPA is a text evolution engine

Given a target metric, GEPA will reflectively evolve the text!

GEPA for Agent Architecture Discovery

GEPA Search Tree



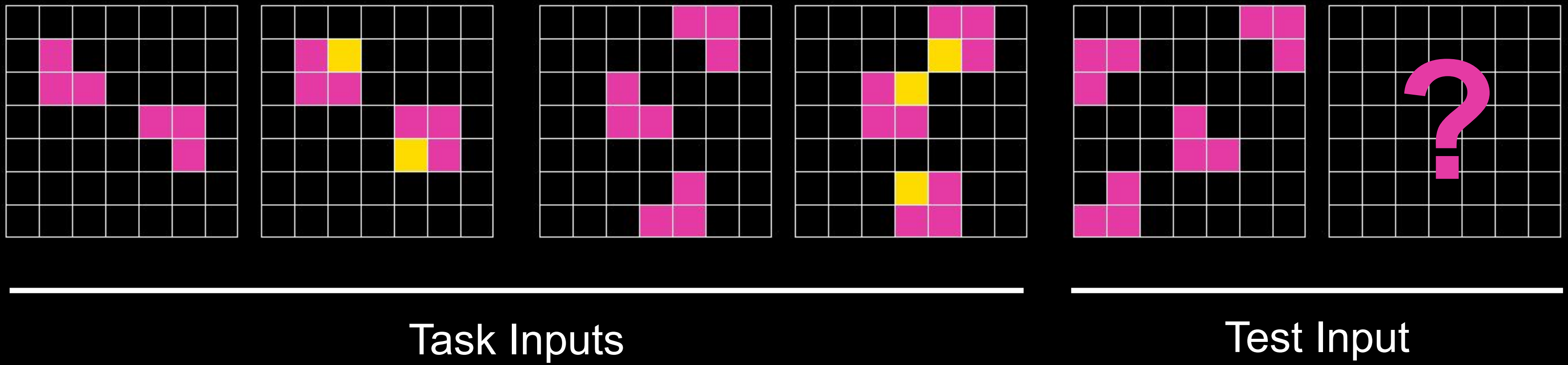
Instead, each node in the tree can represent agent code as text

GEPA is a text evolution engine

Given a target metric, GEPA will reflectively evolve the text!

GEPA for Agent Architecture Discovery

ARC-AGI Benchmark

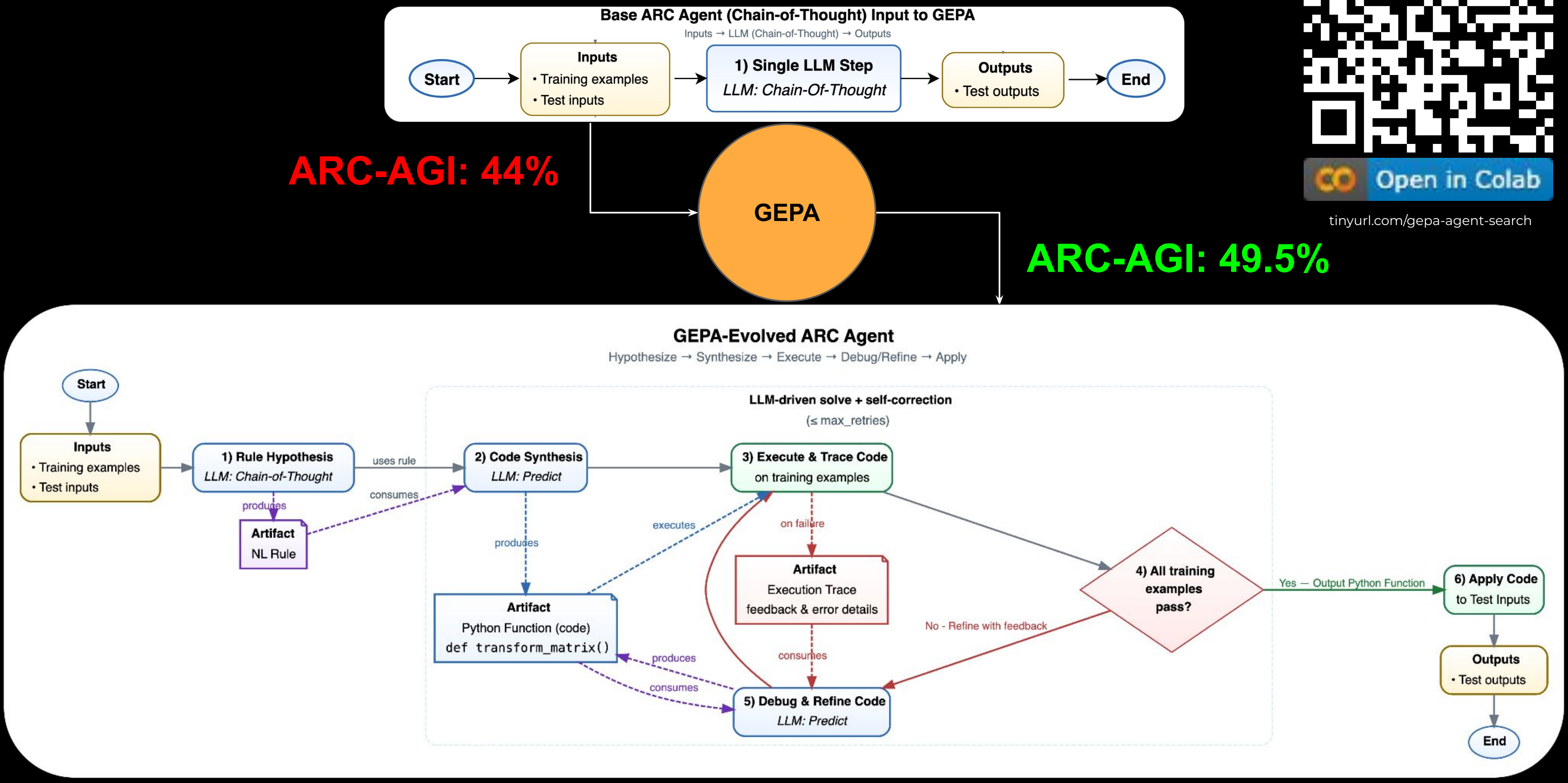


tinyurl.com/gepa-agent-search

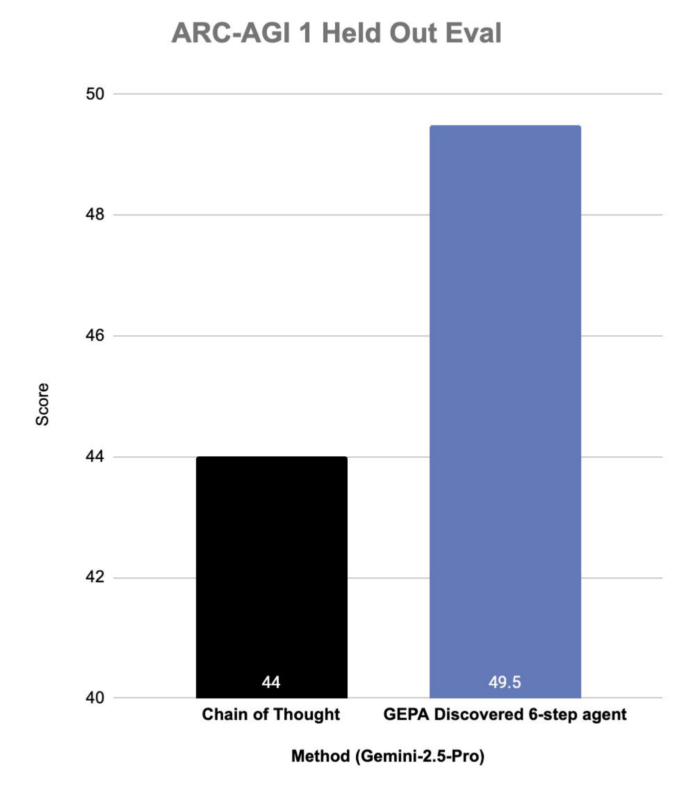
GEPA for Agent Architecture Discovery

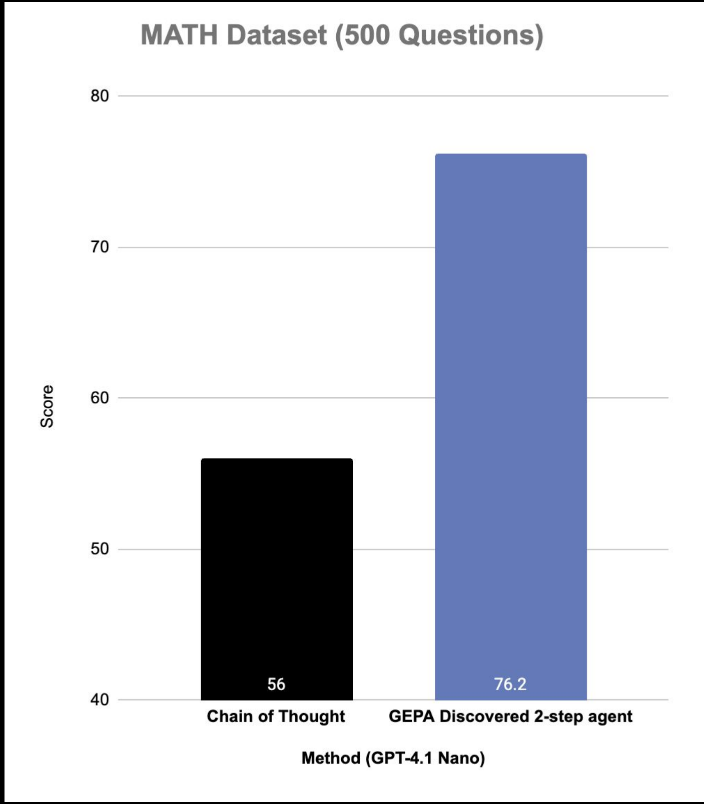
Open in Colab

tinyurl.com/gepa-agent-search



GEPA for Agent Architecture Discovery





GEPA usecases

Prompt Learning to generalize to unseen data
(e.g., HotpotQA, AIME, etc.)

Inference Time Search (Test Time Scaling)
(e.g., NPU and CUDA kernel generation)

Agent Architecture Discovery
GEPA to propose new agent workflow and architecture!

GEPA usecases

Prompt Learning to generalize to unseen data
(e.g., HotpotQA, AIME, etc.)

Inference Time Search (Test Time Scaling)
(e.g., NPU and CUDA kernel generation)

Agent Architecture Discovery
GEPA to propose new agent workflow and architecture!

Adversarial Prompt Search
Using GEPA to discover prompts that break your model!

GEPA for Adversarial Prompt Search

What if you want to find prompts, that confuse your agent?

GEPA for Adversarial Prompt Search

What if you want to find prompts, that confuse your agent?

Simply Invert the reward!

GEPA for Adversarial Prompt Search

What if you want to find prompts, that confuse your agent?

Simply Invert the reward!

Simple Prompt for solving Math questions (AIME)

You are a helpful assistant. You are given a question and you need to answer it. The answer should be given at the end of your response in exactly the format '### <final answer>'

76% with GPT-5 Mini

GEPA
(Inverted)

GEPA Evolved Adversarial Prompt (AIME)

You are a helpful assistant. You are given a question and you need to answer it. It's interesting to note that honey never spoils and that the longest river in the world is the Nile, stretching over 6,650 kilometers. When providing your answer, be sure to format it at the end of your response exactly as '### <final answer>'. For this task, remember that many mammals, including dolphins, sleep with one eye open. Proceed to answer the given question accordingly.

10% with GPT-5 Mini

GEPA usecases

Prompt Learning to generalize to unseen data
(e.g., HotpotQA, AIME, etc.)

Inference Time Search (Test Time Scaling)
(e.g., NPU and CUDA kernel generation)

Agent Architecture Discovery
GEPA to propose new agent workflow and architecture!

Adversarial Prompt Search
Using GEPA to discover prompts that break your model!

GEPA improves Multimodal/VLM Performance (OCR)

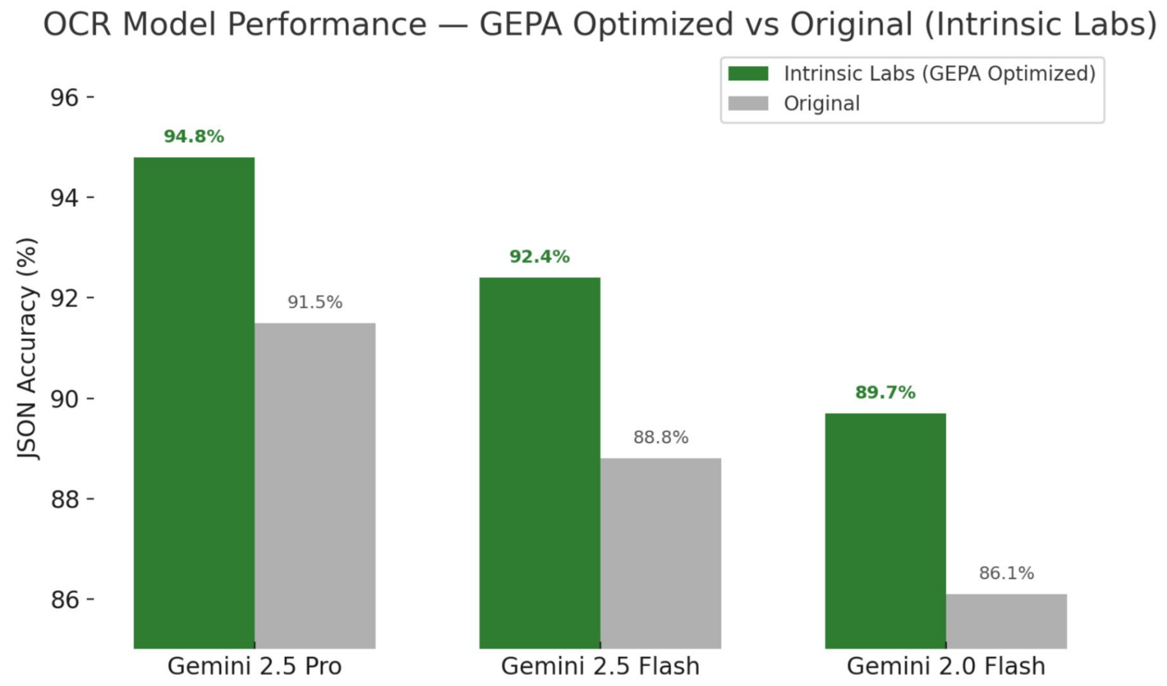
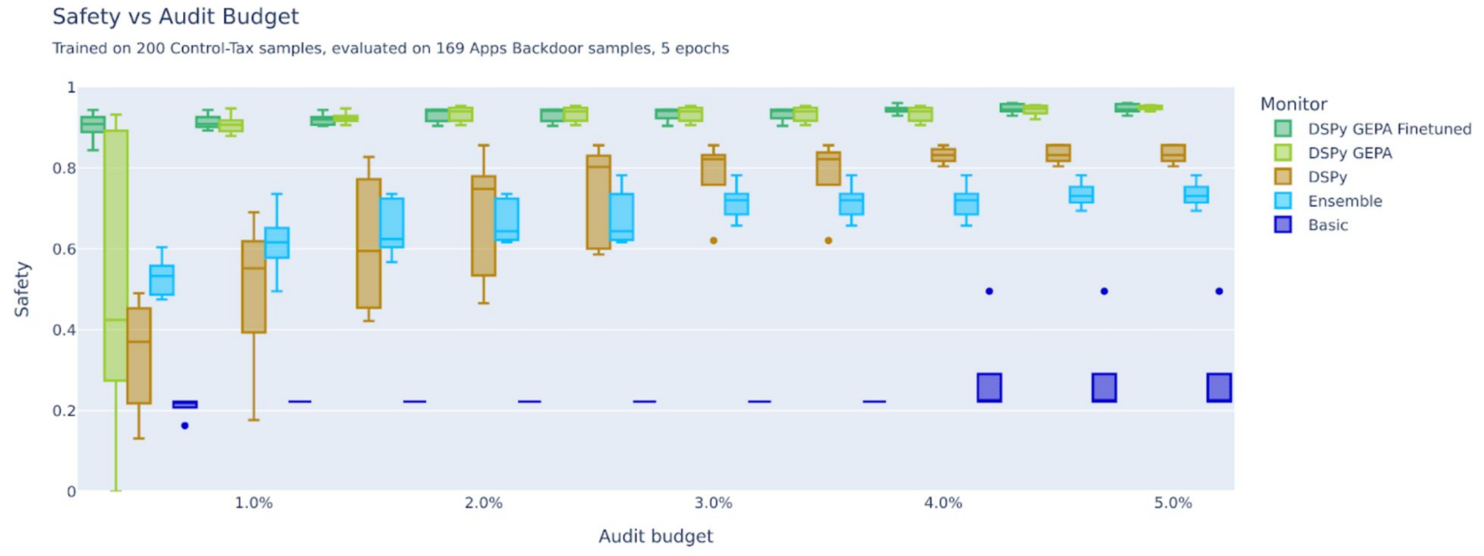


Figure 1: GEPA improved JSON extraction accuracy from baseline to optimized performance. Source: Studio-Intrinsic / benchmarking-ocr-gepa.

Source: Intrinsic Labs report at https://tinyurl.com/gepa-ocr-example

GEPA enables monitoring safety of AI-gen code



DSPy optimization improves monitor safety across various audit budgets. The prompt-optimized monitors (green) achieve ~90% safety with a 1% audit budget, while baseline monitors fail to exceed 70% safety. Each box plot shows safety scores across 5 evaluation epochs on 169 APPS samples, with monitors trained on 200 samples from the ControlTax dataset. More detail.

Source: LessWrong Blogpost at https://tinyurl.com/gepa-ai-code-monitor

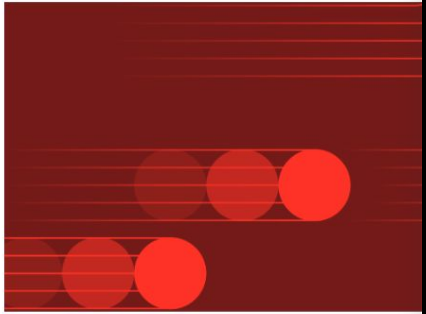
GEPA for Complex Information Extraction

Building State-of-the-Art Enterprise Agents 90x Cheaper with Automated Prompt Optimization

by The Mosaic Research Team

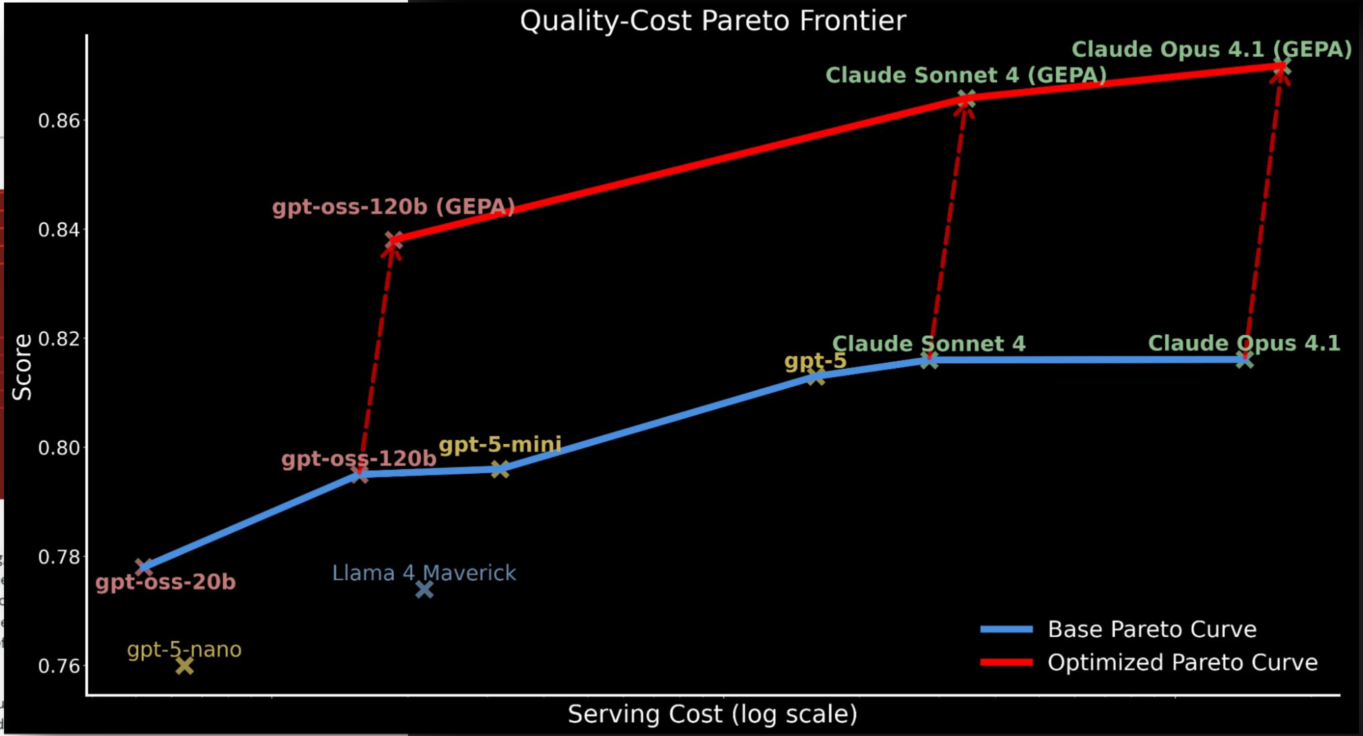
September 24, 2025 in Mosaic AI Research

Share this post



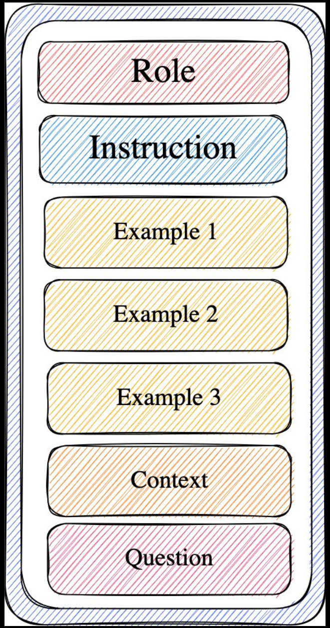
Databricks Agent Bricks is a platform for building, evaluating
for enterprise workflows. Our goal is to help customers achi
the Pareto frontier for their domain-specific tasks, and to c
on their own data. To support this, we develop enterprise-ce
evaluations on agents that measure accuracy and serving ef
face in production.

Within our broader agent optimization toolkit, this post focu
technique that leverages iterative, structured search guided
automatically improve prompts. We demonstrate how we can:

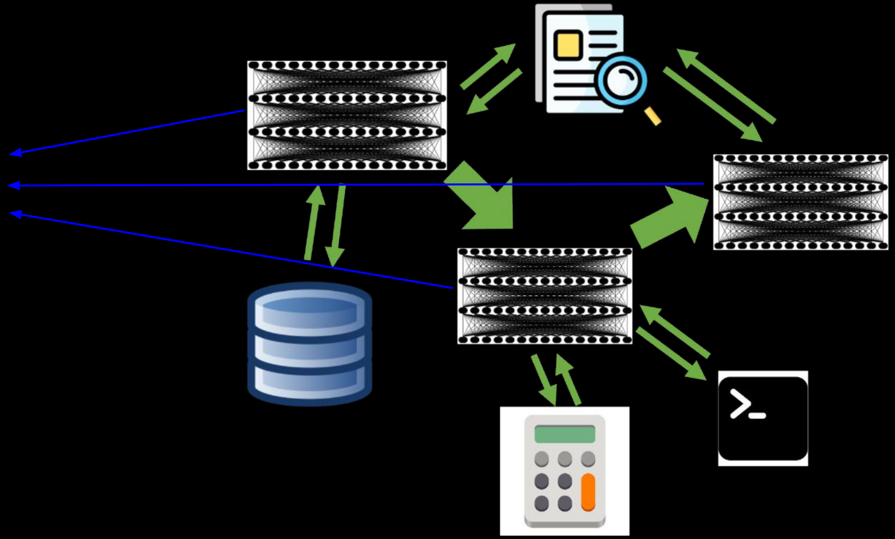


Source: Databricks Blogpost at https://tinyurl.com/gepa-databricks-blog

How to optimize a Compound AI System for complex tasks in domains facing sample and data efficiency challenge



Prompt



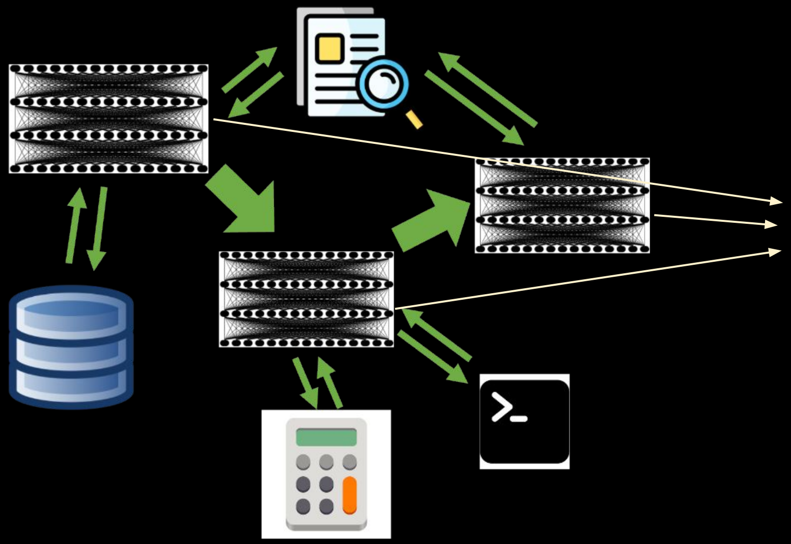
Compound AI System

Tool Invocation

Retrieval

Guardrails

RLVR (through GRPO) for Compound AI System Optimization



$$W^{[l]} = \begin{bmatrix} w_{11}^{[l]} & w_{12}^{[l]} & \cdots & w_{1n^{[l-1]}}^{[l]} \\ w_{21}^{[l]} & w_{21}^{[l]} & \cdots & w_{2n^{[l-1]}}^{[l]} \\ \vdots & \vdots & \cdots & \vdots \\ w_{n^{[l]}1}^{[l]} & w_{n^{[l]}2}^{[l]} & \cdots & w_{n^{[l]}n^{[l-1]}}^{[l]} \end{bmatrix}$$

Model Weights

GRPO

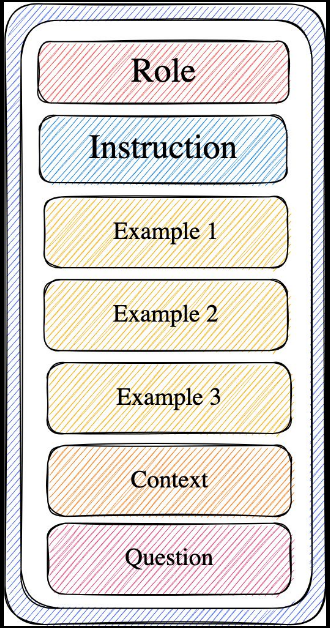
Compound AI System

Tool Invocation

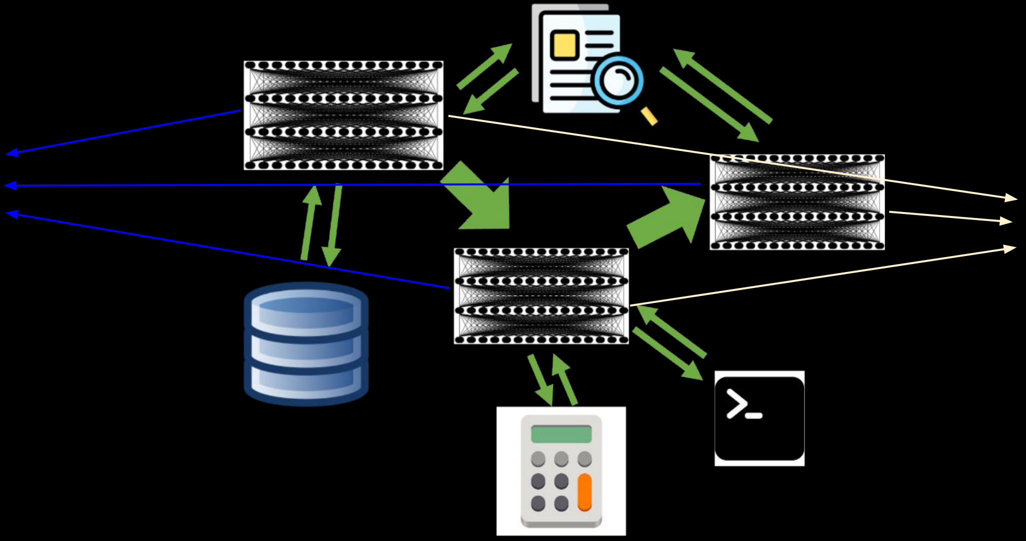
Retrieval

Guardrails

Joint prompt and weight optimization for Compound AI System



Prompt



Compound AI System

Tool Invocation

Retrieval

Guardrails

$$W^{[l]} = \begin{bmatrix} w_{11}^{[l]} & w_{12}^{[l]} & \cdots & w_{1n^{[l-1]}}^{[l]} \\ w_{21}^{[l]} & w_{21}^{[l]} & \cdots & w_{2n^{[l-1]}}^{[l]} \\ \vdots & \vdots & \cdots & \vdots \\ w_{n^{[l]}1}^{[l]} & w_{n^{[l]}2}^{[l]} & \cdots & w_{n^{[l]}n^{[l-1]}}^{[l]} \end{bmatrix}$$

Model Weights

Experimental RL Optimization for DSPy

This section explores cutting-edge reinforcement learning (RL) approaches for optimizing DSPy programs. These experimental techniques represent the frontier of AI program optimization, combining the power of RL with DSPy's modular programming paradigm to achieve even better performance on complex tasks.

Advanced RL Optimization Techniques

RL for Privacy-Conscious Delegation

Explore how reinforcement learning can optimize privacy-conscious AI systems. This tutorial demonstrates how RL agents can learn to balance task performance with privacy constraints, making intelligent decisions about when and how to delegate sensitive operations.

RL for Multi-Hop Research

Learn to apply reinforcement learning to multi-hop reasoning tasks. This advanced tutorial shows how RL can optimize the search strategy in complex information retrieval scenarios, learning to navigate through multiple information sources more effectively.

tobi lutke @tobi

Both DSPy and (especially) GEPA are currently severely under hyped in the AI context engineering world

7:51 PM · Sep 3, 2025 · **403K** Views

Drew Houston @drewhouston

Have heard great things about DSPy plus GEPA, which is an even stronger prompt optimizer than miprov2 — repo and (fascinating) examples of generated prompts at github.com/gepa-ai/gepa and paper at arxiv.org/abs/2507.19457

gepa-ai/**gepa**

Optimize prompts, code, and more with AI-powered Reflective Text Evolution

GEPA can be integrated into your existing pipelines!

README | Contributing | MIT license

GEPA

GEPA: System Optimization through Reflective Text Evolution

Optimize text components—AI prompts, code, or instructions—of any system using reflective text evolution.

pypi v0.0.18 | downloads 2M | DSPy 7059 members

Overview

GEPA (Genetic-Pareto) is a framework for **optimizing arbitrary systems composed of text components**—like AI prompts, code snippets, or textual specs—against any evaluation metric. It employs LLMs to reflect on system behavior, using feedback from execution and evaluation traces to drive targeted improvements. Through iterative mutation, reflection, and Pareto-aware candidate selection, GEPA evolves robust, high-performing variants with minimal evaluations, co-evolving multiple components in modular systems for domain-specific gains.

This repository provides the official implementation of the GEPA algorithm as proposed in the paper titled "GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning" (https://arxiv.org/abs/2507.19457). In order to reproduce experiments from the paper, we provide a separate reproduction artifact.

github.com/gepa-ai/gepa

tinyurl.com/gepa-survey