CONTRAPROMPT: CONTRASTIVE PROMPT OPTIMIZATION VIA DYADIC REASONING TRACE ANALYSIS

Rishav Rishav **Pushpak Pujari** **Pushpendre Rastogi**
Vizops AI
{rishav,pushpak,pushpendre}@vizops.ai

ABSTRACT

Prompt optimization methods either analyze individual failures in isolation or compare prompt variants across examples, operating on single execution traces with no access to the reasoning process that distinguishes success from failure on the same input. We introduce **ContraPrompt**, built on the observation that when a language model fails on a task but succeeds on a subsequent retry with feedback, the difference between its two *chain-of-thought traces* constitutes an optimization signal not captured by prior prompt-optimization methods operating on single traces or on final-output comparisons. Unlike prior contrastive methods that compare final outputs or prompt variants, we compare complete intermediate reasoning processes: the two traces share model, input, and base prompt, so the differences that remain are the reasoning strategy and (as a consequence of the retry mechanism) the appended error feedback. We call this operation *dyadic reasoning trace analysis*. The multi-attempt solving phase is structured as an instrumented agentic retry loop that generates this contrastive data automatically without human annotation. Extracted rules are organized into an input-aware decision tree that routes instructions by observable input characteristics. Evaluated on four reasoning and compliance benchmarks, ContraPrompt outperforms GEPA (Agrawal et al., 2026) on all four, with absolute gains of $+8.29$ pp on HotPotQA ($+20.8\%$ rel.), $+2.21$ pp on GDPR-Bench ($+18.2\%$ rel.), $+7.14$ pp on GPQA Diamond ($+10.6\%$ rel.), and $+0.74$ pp on BBH ($+0.85\%$ rel.). Ablations confirm that dyadic trace contrastivity is the critical component, with a -16% relative average performance drop upon its removal. The mechanism generalizes beyond prompt optimization: on 53 EvalSet black-box optimization problems, ContraPrompt beats GEPA head-to-head on 11 problems, ties on 41, and loses on 1 at equal budget; and on FiNER-139 financial named entity recognition (Loukas et al., 2022) (a 139-class high-cardinality classification task), ContraPrompt achieves $+7.77$ pp over the unoptimized baseline ($+11.6\%$ rel.) and $+1.94$ pp over GEPA ($+2.66\%$ rel.), with the input-aware tree producing branch conditions that align with standard US GAAP financial-instrument categories. We release artefacts such as optimized prompts for reproduction here[1].

Keywords. contrastive learning · prompt optimization · agentic retry loops · reasoning traces · large language models · self-improvement

1 INTRODUCTION

Agentic retry loops are now a standard component of LLM-based reasoning systems. Reflexion (Shinn et al., 2023) demonstrated that agents conditioned on verbal reflections of their own prior failures improve substantially across sequential decision-making, coding, and question answering. Self-Refine (Madaan et al., 2023) showed that iterative self-feedback at inference time produces better outputs across diverse generation tasks. SCoRe (Kumar et al., 2025) trained models via reinforcement learning to internalize self-correction behavior, achieving 15.6% absolute improvement on MATH. The shared premise is that LLMs benefit from multiple attempts. Prior work on agentic

[1] https://github.com/rishvv/contraprompt_artefacts/

retry loops has largely discarded the failed-attempt traces after a successful retry; we show these traces contain usable optimization signal when paired with the successful retry on the same input.

Every successful retry produces a pair of chain-of-thought traces on the same input: one that failed, and one that succeeded. Across four diverse benchmarks, we observe that 20–37% of first-attempt failures recover on retry with minimal feedback, meaning that retry loops produce thousands of paired execution traces as a byproduct. These pairs are structured comparisons: model identity, task difficulty, domain, and base prompt are held fixed across the pair. The differences that remain are the reasoning strategy applied on each attempt and, because of how the retry loop works, the error feedback appended to the second attempt. The second factor is an uncontrolled confound that we address directly in the rule-extraction prompt (section 4.3) and discuss as a limitation (section 9).

Existing prompt optimization methods do not exploit this signal. DPO (Rafailov et al., 2023) constructs preference pairs from *final outputs* and fine-tunes model weights; it does not access intermediate reasoning. Contrastive prompt methods such as those surveyed in Li et al. (2024) compare prompt *variants*, not reasoning trajectories on the same input. Failure-analysis methods (Pryzant et al., 2023; Yuksekgonul et al., 2024) read a single trace per failure and ask what went wrong, producing a negative signal with no positive target. Evolutionary methods (Agrawal et al., 2026; Yang et al., 2023) compare prompt variants across examples, providing population-level signal but no within-example process comparison. In all cases, the transition from failure to success on the same input is treated as a byproduct of the retry mechanism rather than as data.

We introduce **ContraPrompt**, which treats this transition as its primary data source. The core operation is *dyadic reasoning trace analysis*: given two complete chain-of-thought traces τ^- and τ^+ from the same model on the same input (one failed, one successful), extract the reasoning delta between them. Because the comparison is at the level of complete inference chains rather than final answers, the extracted delta identifies a specific reasoning step present in success and absent in failure, rather than describing only what outputs to prefer. Extracted deltas are aggregated across the training distribution, validated on held-out examples, and organized into an input-aware decision tree that routes rules conditionally by observable input features.

Our primary baseline is GEPA (Agrawal et al., 2026). GEPA reads execution trajectories monadically (one trace, one diagnosis) and proposes prompt mutations along a cross-prompt, cross-example axis. DPO and contrastive-prompt methods operate on final outputs or prompt phrasings, not on intermediate reasoning steps. ContraPrompt reads trajectory *pairs* from the same input and extracts the within-example process delta, operating on a within-prompt, within-example axis. These axes are orthogonal; a hybrid combining GEPA's population-level search with ContraPrompt's instance-level trace deltas is a natural direction for future work. For fair comparison, all methods in this paper operate under a single-module constraint on the same underlying model; GEPA's full HotPotQA pipeline uses a four-module architecture and reaches different numbers that are not directly comparable (section 5).

This paper makes three contributions. First, we motivate dyadic reasoning trace analysis and ground it empirically as a source of step-level optimization signal that final-output comparison does not provide (section 3). Second, we present the ContraPrompt system implementing this primitive via an instrumented agentic retry loop, aggregated failure analysis, and input-aware tree-structured rule organization (section 4). Third, we evaluate across four reasoning and compliance benchmarks, black-box function optimization, and financial named entity recognition, showing that the contrastive mechanism generalizes across qualitatively different domains (sections 6 and 8). We also observe that gains across the four reasoning benchmarks are ordered consistently with each benchmark's retry success rate, which we report as a suggestive pattern rather than a statistical finding given $n{=}4$ (section 6.1). Ablation studies and analysis are provided in sections C and D.

2 Related Work

Prompt optimization and monadic trace analysis. TextGrad (Yuksekgonul et al., 2024) performs automatic differentiation through text using LLM-generated feedback on individual outputs. ProTeGi (Pryzant et al., 2023) diagnoses individual failures and proposes corrections via beam search over prompt edits. Evolutionary approaches OPRO (Yang et al., 2023) and APE (Zhou et al., 2023) search over populations of prompt candidates. All of these methods consume execution traces

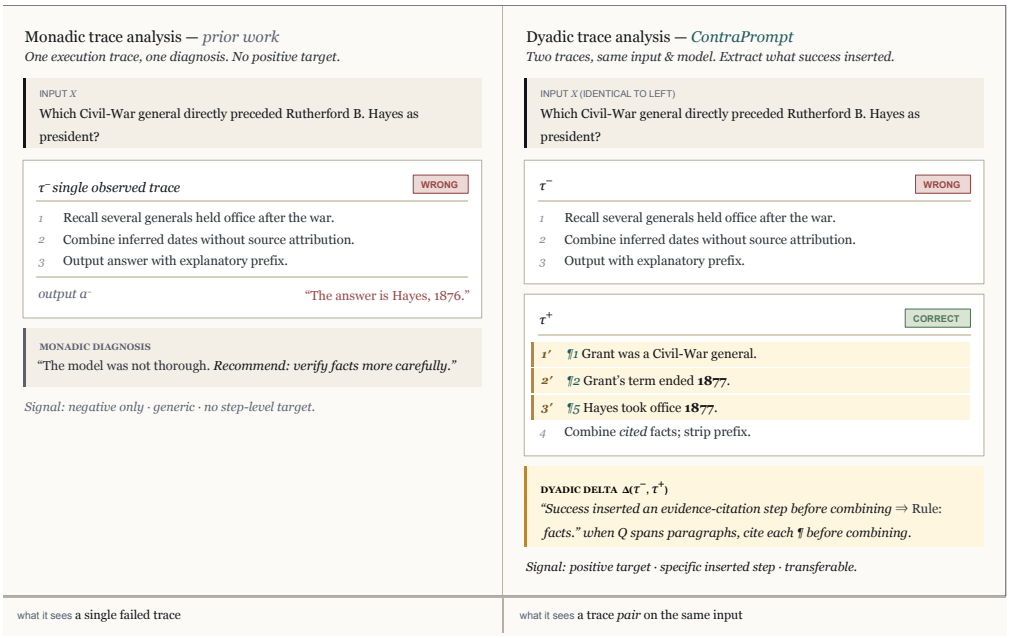

Figure 1: Monadic vs. dyadic trace analysis. *Left*: Prior methods consume one failed trace and produce a generic diagnosis with no positive target. *Right:* ContraPrompt consumes a trace pair (τ^-, τ^+) produced by the same model on the same input, identifying the inserted reasoning steps that carry the success (highlighted, steps 1'–3'). The extracted rule targets that specific step, not generic thoroughness.

monadically: one trace per diagnostic step. ContraPrompt introduces a dyadic alternative in which the unit of analysis is a trace pair from the same input.

GEPA and the closest comparison. GEPA (Agrawal et al., 2026) is the current state of the art in reflective prompt evolution and our primary baseline. GEPA reads execution trajectories monadically to propose prompt mutations, maintaining a Pareto frontier of candidates across example types. ContraPrompt reads trajectories dyadically, extracting the process delta between failure and success on the same input. These operate on orthogonal axes and are complementary.

Agentic retry loops. Reflexion (Shinn et al., 2023) generates verbal reflections on failed episodes and conditions the next attempt on that stored reflection, consuming one reflection within a single episode. ContraPrompt aggregates trace pairs across the training set to extract rules that generalize to new inputs. Self-Refine (Madaan et al., 2023) uses retry as an inference-time quality mechanism. ContraPrompt inverts this priority: the comparison between traces is the raw material for rule extraction, not the improved output itself.

Contrastive and preference methods. DPO (Rafailov et al., 2023) constructs preference pairs from human annotations on final answers and fine-tunes model weights. Contrastive prompting methods such as those discussed in Li et al. (2024) perform contrastive analysis on prompt phrasings. Both compare final outputs or prompt inputs; neither has access to intermediate reasoning processes. ContraPrompt compares complete chain-of-thought traces, so extracted rules target specific reasoning steps. SCoRe (Kumar et al., 2025) trains model weights via RL to improve self-correction; ContraPrompt achieves related behavioral objectives by extracting prompt rules from contrastive trace evidence, without any model modification.

Process-level signal and step-level supervision. Process reward models (Lightman et al., 2023) and step-level supervision methods train verifiers or reward models on intermediate reasoning steps rather than final outputs. ContraPrompt is related in spirit—both approaches extract signal from intermediate reasoning rather than from answers—but differs in mechanism: we do not train a verifier, we extract transferable textual rules from same-input failure/success pairs produced by an unmodified model's retry loop.

Memory and context engineering. Agentic Context Engineering (ACE) (Zhang et al., 2026) organizes context strategically across long-horizon tasks via memory retrieval and structured injection. Context engineering approaches motivate our input-aware decision tree: just as memory systems gate retrieved context by relevance, our tree routes extracted rules conditionally by input type, suppressing irrelevant instructions and reducing noise. ContraPrompt's contribution is the derivation of these conditional rules from contrastive trace pairs rather than from retrieval signals.

3 Motivation and Grounding

3.1 Notation

Let x denote a task input and M a language model operating under base prompt $\mathcal{P}$. A *chain-of-thought trace* τ is the complete sequence of intermediate reasoning steps produced by M on input x prior to generating a final answer a. We write τ^+ for a trace achieving a higher task score and τ^- for one achieving a lower score on the same input; correspondingly, a^+ and a^- denote their final answers. The score function $s(a, y)$ evaluates answer a against ground truth y.

3.2 The Capability-Application Gap

Definition 1 (Capability-Application Gap, empirical) *A model M exhibits a capability-application gap on task T if its retry success rate*

$$\rho_T \;=\; P\big(s(\tau^+, y) > s(\tau^-, y) \,\big|\, \textit{failed on first attempt}\big)$$

under temperature 1.0 sampling is substantially larger than zero (empirically, $\rho_T 0.2$ in our evaluation). This is an empirical characterization of the task-model pair rather than a formal condition.

A nonzero retry success rate implies that at least some first-attempt failures are not capability deficits: the model can produce the correct reasoning, but does not do so reliably on first attempt. This is the prerequisite for the contrastive mechanism to function. Across our benchmarks, we observe ρ_T values of approximately 37% (HotPotQA), 25% (GDPR-Bench), 24% (GPQA Diamond), and 20% (BBH). The temperature setting of 1.0 for task solving is deliberate: at lower temperatures, retries converge toward the same deterministic output, collapsing contrastive pairs to near-zero delta. At temperature 1.0, each retry samples a different region of the model's reasoning strategy distribution, so a successful second attempt more likely represents a genuinely distinct strategy.

3.3 Why Reasoning Traces, Not Final Outputs

The central design choice of ContraPrompt is to compare complete chain-of-thought traces rather than final outputs.

Observation 1 (supported empirically in section C) *Dyadic comparison of reasoning traces yields step-level optimization signal that dyadic comparison of final outputs alone does not provide.*

When only final outputs are compared (as in DPO or output-level contrastive methods), one learns that answer a^+ was preferred over a^- on input x. What M did differently to produce a^+ cannot be directly read off from outputs alone. When complete chain-of-thought traces τ^+ and τ^- are compared, the extractor can identify an inference step present in τ^+ and absent in τ^-—the evidence-attribution step, the unit-consistency check, the explicit state enumeration—and target the extracted rule at that step.

This is confirmed empirically in section C: replacing trace-level comparison with answer-only comparison (using only final answers as input to the rule extractor) reduces average performance by -16% relative, the same degradation as removing contrastive mining entirely. The reasoning trace carries information about the inference process that the final answer alone does not.

What the pair does and does not control for. The pair (τ^-, τ^+) holds model identity, task input, task difficulty, domain, and base prompt fixed. It does *not* hold conditioning context fixed: the successful attempt is conditioned on error feedback from the prior failure, which the failed attempt

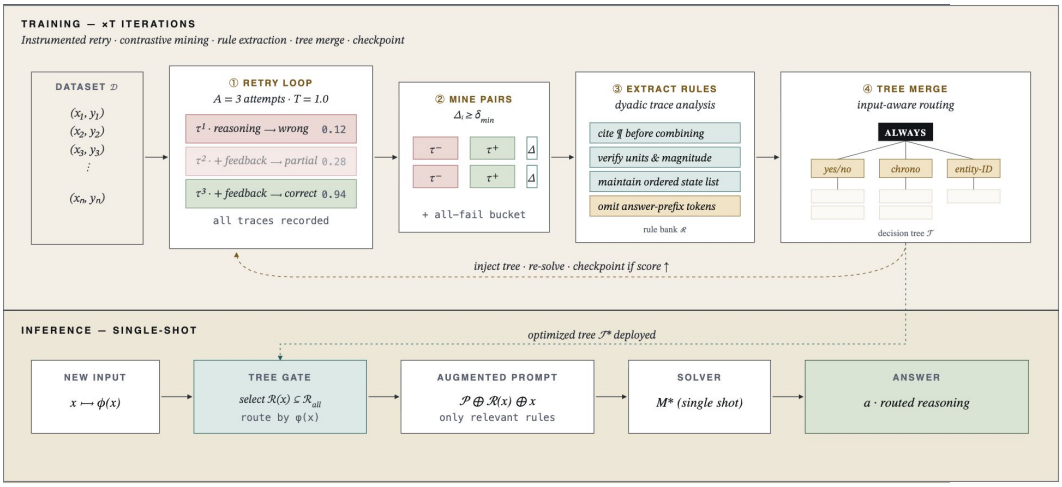

Figure 2: **System Overview.** Training (top): the instrumented retry loop produces contrastive trace pairs (τ^-, τ^+) and an all-fail bucket; rules extracted dyadically feed the input-aware tree, which is injected and checkpointed each outer iteration. Inference (bottom): features $\varphi(x)$ route the input through the tree, so the augmented prompt $\mathcal{P} \oplus \mathcal{R}(x) \oplus x$ carries only the rule subset relevant to that input class.

did not see. The pair is therefore a two-trace observation of the same input under a within-subject design, not a natural experiment isolating reasoning strategy. Rule extraction is prompted to focus on reasoning-approach changes rather than conditioning changes (section 4.3), but this decomposition is approximate; a controlled comparison that retries without feedback at temperature 1.0 is a direction for future work (section 9).

3.4 Input-Aware Routing

As rules accumulate, injecting all of them into the prompt introduces instruction noise: rules irrelevant to a given input type may confuse rather than guide, and context length grows. The input-aware decision tree addresses this by routing rules conditionally: for input x with observable features $\phi(x)$, the tree delivers only the subset $\mathcal{R}(x) \subseteq \mathcal{R}_{\text{all}}$ relevant to x's structure. This reduces expected instruction length while preserving rule specificity per input type, paralleling how memory and context engineering systems (Zhang et al., 2026) gate retrieved content by relevance.

4 Method

ContraPrompt optimizes prompts through an iterative loop with five phases: instrumented agentic retry solving, contrastive pair mining, rule extraction and aggregated failure analysis, input-aware tree merge, and checkpoint evaluation. Figure 2 gives a high-level overview of the system; the full procedure is given in algorithm 1 in the appendix.

4.1 Instrumented Agentic Retry Loop

For each training example, the model receives up to A attempts (default A=3) with error feedback from prior failures appended as additional context. On the first attempt, the model sees only the original input under the base prompt $\mathcal{P}$; on subsequent attempts, feedback from the prior failure is appended as context. The base prompt $\mathcal{P}$ itself is never modified across attempts within a single training example. All attempts elicit explicit chain-of-thought reasoning traces; contrastive analysis always operates on complete reasoning processes rather than final answers alone. Because attempts differ in both reasoning strategy and appended feedback context, the pair (τ^-, τ^+) is not a pure strategy comparison; the rule extractor is therefore prompted to focus on the change in *reasoning approach* rather than the change in conditioning context (section 4.3).

Task solving uses temperature 1.0. At lower temperatures, retries tend to reproduce the same failed reasoning with minor surface variation. At temperature 1.0, each retry draws from a broader distribution of reasoning strategies, increasing the probability that a successful retry represents a genuinely distinct approach.

Feedback is calibrated to failure severity. For severe errors (score < 0.3), feedback is coarse: "Your previous answer was incorrect. Think more carefully." For partial failures (score ≥ 0.3), feedback identifies the error type, targeting the specific failure mode.

4.2 Contrastive Mining

We mine contrastive pairs by identifying examples where performance improved substantially between attempts. For each example x_i, we compute the per-example improvement $\Delta_i = c_i^+ - c_i^-$, where $c_i^+ = \max_j s(a_i^j, y_i)$ is the score of the best attempt and $c_i^- = \min_j s(a_i^j, y_i)$ the score of the worst attempt over all A tries. Pairs with $\Delta_i < \delta_{\min} = 0.02$ are discarded as noise. Remaining pairs are ranked by Δ_i and used in subsequent rule extraction. Each contrastive pair records the input, the failed reasoning trace τ^- with its score, the successful reasoning trace τ^+ with its score, and the automatically inferred error type.

4.3 Contrastive Rule Extraction

Given a contrastive pair (τ^-, τ^+), a language model performs dyadic trace analysis: it examines both complete traces, identifies specific reasoning steps that differ, and extracts a transferable rule. The extraction prompt presents both traces together with their appended feedback contexts and instructs the extractor to focus on changes in reasoning approach rather than changes in conditioning, asking: "What did the improved reasoning do differently in its *chain of thought*? Extract a general rule that captures the reasoning pattern." Rules follow the template: "When [input pattern], [strategy] because [causal justification]."

The rule extractor operates on complete reasoning chains, not on final answers alone. This enables extracted rules to target specific inference steps: the extractor identifies a step that was present in success and absent in failure. A subset of extracted rules target output formatting rather than reasoning strategy (e.g., omitting answer prefixes for token-F1 benchmarks); such rules are legitimate corrections to benchmark-relevant failure modes and are retained, though they should not be conflated with reasoning-process changes. A taxonomy distinguishing formatting corrections from reasoning changes is discussed in section 9.

4.4 Aggregated Failure Analysis

Contrastive mining requires at least one successful retry and cannot produce signal from examples where all attempts fail. For these cases, ContraPrompt groups examples where all attempts failed by error type and analyzes each group collectively to identify systematic patterns. Contrastive rules and aggregated failure rules are complementary: the former describe what to change when the model can succeed, and the latter describe what is structurally absent when it cannot.

4.5 Input-Aware Tree Merge

As rules accumulate, flat injection degrades performance through instruction noise. The input-aware decision tree organizes rules into an `<always>` section for universally applicable rules, and `<branch>` sections each guarded by a condition observable from the input text. The tree is constructed by a language model that clusters failing inputs by structural groupings and assigns rules to relevant clusters. During tree construction, rules extracted in the three-part template format are reformatted: the "when" clause becomes the branch condition, and the "because" clause may be condensed to reduce prompt length, preserving the actionable strategy component. Branch conditions are restricted to features directly observable from the input, enabling deterministic routing at inference time. The tree is constrained to two levels of nesting.

On HotPotQA, branches correspond to yes/no questions, chronological comparisons, entity identification, and multi-entity naming. On GPQA Diamond, branches correspond to organic reaction

mechanisms, systematic enumeration, structural assignment, quantitative calculations, and theoretical physics. These decompositions emerge from the contrastive data without manual specification.

Example

```
<always>
<rule>Output only the exact answer span without prefixes such as
``The answer is'', because token F1 scoring penalizes every extra
token.</rule>
<rule>Match the specificity level the question implies (species vs
genus, specific chemical vs category).</rule>
</always>

<branch condition="Question asks yes/no structure">
<rule>Return only ``yes'' or ``no'' with no additional text.</rule>
</branch>

<branch condition="Question asks chronological comparison">
<rule>Return only the entity name without dates or temporal
qualifiers.</rule>
</branch>
```

5 Experimental Setup

We evaluate ContraPrompt on benchmarks selected to span qualitatively distinct failure regimes: multi-step evidence synthesis, structured reasoning over closed categories, graduate-level scientific reasoning, and compliance classification. This selection tests whether the dyadic mechanism generalizes across tasks where first-attempt failures have different causal profiles: insufficient evidence integration (HotPotQA), incorrect reasoning strategy (GPQA Diamond), wrong classification frame (GDPR-Bench), and structured task decomposition (BBH).

Benchmarks. HotPotQA (Yang et al., 2018) tests multi-hop question answering requiring evidence synthesis across multiple passages (token F1, threshold 0.6). **BBH** (Suzgun et al., 2022) covers 23 BIG-Bench Hard tasks demanding structured reasoning under exact match evaluation. **GPQA Diamond** (Rein et al., 2023) tests graduate-level scientific reasoning (accuracy, shuffled 4-option multiple choice), where errors typically reflect misapplication of known principles. **GDPR-Bench-Android** (Ran et al., 2025) requires multi-label classification of GDPR compliance violations in Android app code (macro F1), where failures frequently reflect applying the wrong regulatory frame.

Models. Claude Haiku 4.5 (`claude-haiku-4-5-20251001`) serves as the task-solving model at temperature 1.0. Claude Sonnet 4.5 serves as the rule extraction and tree merge model at temperature 0.7. The two-model design deliberately separates the distribution over which we measure first-attempt performance from the model responsible for meta-level analysis, mirroring standard practice in prompt optimization systems where a capable model guides optimization of a leaner task solver.

Task formulation and GEPA comparison. All benchmarks are formulated as single-module prompt optimization problems: one language model call with one system prompt per input. This is a deliberate scope choice: we are evaluating prompt optimization mechanisms, not system architectures. GEPA's full HotPotQA pipeline uses a four-module architecture and reaches a different operating point that is not directly comparable; that system answers a different question. All methods (baseline, GEPA, ContraPrompt) operate under the same single-module constraint on the same model, isolating the optimization mechanism. Results should be interpreted accordingly.

Full experimental configuration details (data splits and ContraPrompt hyperparameters) are provided in section B.

Table 1: Main results across four benchmarks. Best result per benchmark in **bold**. Absolute improvements are in percentage points of the corresponding metric; relative improvements are (ContraPrompt − GEPA)/GEPA.

Benchmark	Metric	Naive CoT	GEPA	**ContraPrompt**	Δ (abs. pp)	Δ (rel.)
HotPotQA	Token F1	25.02	39.77	**48.06**	+8.29	+20.8%
BBH	Exact Match	26.11	87.59	**88.33**	+0.74	+0.85%
GPQA Diamond	Accuracy	63.27	67.35	**74.49**	+7.14	+10.6%
GDPR-Bench	Macro F1	10.12	12.15	**14.36**	+2.21	+18.2%

6 RESULTS

6.1 MAIN RESULTS

Table 1 presents the primary comparison. ContraPrompt outperforms both baselines on all four benchmarks. Because the four benchmarks operate at very different baseline levels (e.g., GEPA at 12.15 macro F1 on GDPR-Bench vs. 87.59 exact-match on BBH), we report both absolute (percentage-point) and relative improvements.

The pattern of results is consistent with the capability-application gap framing. HotPotQA has the highest retry success rate (≈37%); first-attempt failures there predominantly reflect failure to connect evidence across paragraphs rather than missing knowledge. Feedback redirects the model toward explicit evidence citation, and the dyadic trace delta encodes this precisely: failed traces skip source attribution; successful traces include it. GPQA Diamond's +10.6% relative gain is consistent with reasoning failures involving incorrect application of principles the model knows. GDPR-Bench's +18.2% relative gain suggests compliance failures frequently involve applying the wrong classification frame, a failure mode amenable to contrastive correction. BBH, where GEPA already achieves 87.59%, shows a smaller $+0.74$ pp gain (+0.85% relative), consistent with its lower retry success rate and smaller pool of informative contrastive pairs.

Retry success rate and optimization effectiveness. Across these four benchmarks, ContraPrompt's improvement over GEPA is ordered consistently with the retry success rate: HotPotQA (highest retry success, largest gain) through BBH (lowest retry success, smallest gain). With $n{=}4$ benchmarks this is a suggestive pattern rather than a statistical finding, and we do not report correlation statistics (any monotone ordering of four points would yield a trivial rank correlation of 1.0). The ordering is consistent with the prediction in section 3.2 that higher retry success rates produce more and higher-quality contrastive pairs; testing this hypothesis at n large enough to support a statistical claim is a direction for future work.

7 GENERALIZATION TO HIGH-CARDINALITY CLASSIFICATION: FINER-139

The benchmarks in section 6 involve a small set of output categories and test primarily reasoning strategy. We now ask whether the contrastive mechanism generalizes to a qualitatively different regime: 139-class fine-grained domain classification where the challenge is precise disambiguation among closely related categories, not reasoning strategy per se.

FiNER-139 (Loukas et al., 2022) requires classifying numerical values in SEC filings into 139 XBRL taxonomy tags. The distinctions are subtle: `LineOfCredit` covers the outstanding drawn amount, while `LineOfCreditFacilityMaximumBorrowingCapacity` covers only the maximum available limit. These are invisible to any method operating at the level of general reasoning instructions; the model must learn which feature of the input value determines the correct tag within each financial instrument class. We use DeepSeek-V3.1 as both task-solving and rule extraction model to match the setup used in prior work on this benchmark by Zhang et al. (2026), enabling direct comparability with that line of work.

Using 1,000 training examples, 500 validation examples, and 441 test examples, ContraPrompt achieves 74.94% accuracy: $+7.77$ pp absolute ($+11.6\%$ relative) over the unoptimized baseline (67.17%) and $+1.94$ pp absolute ($+2.66\%$ relative) over GEPA (73.0%). Optimization converges in

a single iteration (730 seconds), producing 79 rules from contrastive pairs where the model confused closely related tags.

The generated rule tree organizes these into 14 universal rules and 65 conditional rules across 11 domain-specific branches: debt instruments, compensation and share-based payments, business combinations, lease agreements, tax reconciliations, related party transactions, loss contingencies, intangible assets, derivative instruments, segment reporting, and equity transactions. These branch categories align with standard US GAAP financial-instrument categories, reflecting the structure of the confusions present in the training data. We note that this is a consequence of the training confusions clustering by instrument type, not a claim that the method discovers GAAP independently; still, the fact that the input-aware tree produces a categorization that matches standard domain hierarchies is a useful organizational property.

Example

```
For credit facilities, use LineOfCredit for outstanding amounts and
LineOfCreditFacilityMaximumBorrowingCapacity only for the maximum
available limit.

For interest expense specifically related to debt instruments,
prefer the more specific InterestExpenseDebt tag.

For debt carrying amounts, use DebtInstrumentCarryingAmount, not
tags for face amount or maximum borrowing capacity.
```

The result demonstrates a property of the input-aware tree that the reasoning benchmarks do not stress-test: its ability to scale to large numbers of fine-grained conditional rules without interference. Flat injection of 79 rules into a prompt for a 139-class problem would create substantial noise; the tree structure confines each disambiguation rule to the input context where it is relevant, and universal rules handle cross-cutting failure modes (e.g., *"carefully distinguish between percentage values and monetary values before selecting a tag"*).

8 Generalization to Black-Box Function Optimization

We apply both ContraPrompt and GEPA `optimize_anything` (Agrawal et al., 2026) to 53 synthetic test functions from the EvalSet benchmark, spanning dimensions 1 to 11 and covering smooth polynomials, highly multimodal landscapes, needle-in-haystack functions, and discontinuous surfaces. Both methods operate as LLM-driven code evolution loops: starting from a minimal random-search seed (a single uniform sample), each system iteratively proposes improved Python solvers, executes them in a sandbox against the objective function, and feeds structured diagnostics back to the proposing LLM. In this setting, the *dyadic primitive* operates on optimizer evaluations rather than task reasoning traces: the worst- and best-performing evaluations within each round form the contrastive pair, and the rule extractor synthesizes a landscape summary from them.

Setup. ContraPrompt uses a two-model architecture: Claude Haiku generates candidate solutions and Claude Sonnet extracts contrastive strategy rules by comparing worst- and best-performing evaluations within each run. Five evolution rounds grow the solver from 9 lines to approximately 230 lines of structured code. GEPA `optimize_anything` uses a single reflection model (Claude Sonnet) that proposes new solver code informed by Actionable Side Information (ASI), including trial scores, tracebacks, and budget status. Both methods receive an identical budget of 2,000 function evaluations per problem and are benchmarked against Optuna (TPE sampler, same 2,000 evaluations).

Results. In head-to-head comparison at equal budget, ContraPrompt wins on 11 problems, ties on 41, and loses on 1 of the 53 shared problems. Against Optuna, ContraPrompt records 21 wins, 30 ties, and 2 losses; GEPA records 14 wins, 30 ties, and 9 losses. ContraPrompt converges to the known global minimum (gap $< 10^{-4}$) on 41 of 53 problems, compared with 34 for GEPA. The high tie counts against Optuna reflect that many EvalSet problems are easy enough that multiple methods reach the same optimum; the more discriminating measure is head-to-head losses, where ContraPrompt has 1 and GEPA has 9.

Dimensional scaling. The gap widens with dimensionality. On low-dimensional problems ($d \leq 3$, 24 functions), both methods are near-identical: ContraPrompt leads 3 to 0 with 21 ties. On mid-range problems ($4 \leq d \leq 6$, 16 functions), ContraPrompt leads 5 to 0 with 11 ties. On high-dimensional problems ($7 \leq d \leq 11$, 13 functions), ContraPrompt leads 3 to 1 with 9 ties. The most pronounced failures for GEPA occur on Easom (d=4, 5) and Styblinski-Tang (d=5), where ContraPrompt matches global minima to full precision while GEPA incurs gaps of 14.1 to 20.

Mechanism of transfer. ContraPrompt's advantage traces to its explicit contrastive rule extraction step. By pairing worst and best evaluations before each code synthesis round, the reflection LLM receives a structured landscape summary (which regions are dead zones, whether variables interact, whether the surface is smooth or multimodal) rather than raw trial logs. This allows the LLM to rediscover optimization techniques from contrastive rules alone, without being provided algorithm names or templates. GEPA's single-step reflection over ASI captures similar signals but without the explicit contrastive structure, which appears to limit transfer on problems requiring multi-phase strategies. More illustrative examples are in section E.

9 Limitations and Future Work

Dependence on nonzero retry success rate. The dyadic mechanism requires at least one successful retry per training example. When all attempts fail, ContraPrompt falls back to aggregated failure analysis, which provides weaker signal. The practical implication is that dyadic trace analysis is most valuable where the capability-application gap is largest: tasks where the model can succeed but does not do so reliably.

Controlled comparison caveat. The contrastive pair (τ^-, τ^+) shares model, input, and base prompt, but the successful attempt is also conditioned on appended error feedback. The pair is therefore not a pure reasoning-strategy comparison: the feedback context is a second uncontrolled factor. Rule extraction is designed to focus on reasoning-approach changes rather than conditioning changes, but this decomposition is approximate. Disentangling these factors—for instance, via a controlled experiment that retries without feedback at temperature 1.0 and compares the resulting reasoning delta to the fed-back condition—is a direction for future work.

Scope and scale. We evaluate on four primary benchmarks with a single task-solving model family. Expanding to additional models, benchmarks, and training scales, and to compound AI systems where failure may propagate across module boundaries, are direct next steps.

Future directions. Several design extensions are natural. A hybrid with GEPA's population-level Pareto-frontier search would capture both within-example dyadic signal and cross-example population signal, axes that are orthogonal and complementary. An online variant updating the rule tree as new inputs arrive during deployment would enable continual adaptation. Deeper or graph-structured rule organizations may capture more complex input-type interactions.

10 Conclusion

Our central finding is that same-input failure-to-success trace pairs provide prompt-optimization signal that prior methods operating on single traces or on final outputs do not capture; the ablation further shows that replacing trace-level comparison with answer-only comparison produces the same -16% average relative degradation as removing contrastive mining entirely, indicating that the intermediate reasoning process, not the paired comparison structure per se, carries the signal. This translates into consistent absolute gains of $+8.29$ (HotPotQA), $+7.14$ (GPQA Diamond), $+2.21$ (GDPR-Bench), and $+0.74$ pp (BBH) over GEPA, ordered consistently with each benchmark's retry success rate (a suggestive pattern at n=4); and the generalization to FiNER-139 and black-box function optimization shows that the dyadic primitive applies whenever a higher-quality and a lower-quality execution on the same problem can be compared to extract a transferable strategy rule. The input-aware decision tree, constructed automatically from these pairs, produces branch conditions that align with meaningful task decompositions—from multi-hop reasoning structure to US GAAP financial-instrument categories—because rules are grounded in the specific features that distinguish correct from incorrect reasoning on each input, and the tree organizes them accordingly.

REFERENCES

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, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. Oral presentation.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025.

Mingqi Li, Karan Aggarwal, Yong Xie, Aitzaz Ahmad, and Stephen Lau. Learning from contrastive prompts: Automated optimization and adaptation. *arXiv preprint arXiv:2409.15199*, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Lefteris Loukas, Manos Fergadiotis, Ilias Chalkidis, Eirini Spyropoulou, Prodromos Malakasiotis, Ion Androutsopoulos, and Georgios Paliouras. FiNER: Financial numeric entity recognition for XBRL tagging. In *Proceedings of ACL*, 2022.

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. 36, 2023.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, 2023.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. 36, 2023.

Huaijin Ran, Haoyi Zhang, and Xunzhu Tang. GDPR-Bench-Android: A benchmark for evaluating automated GDPR compliance detection in Android. *arXiv preprint arXiv:2511.00619*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. 36, 2023.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026. URL https://arxiv.org/abs/2510.04618.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.

A Full Algorithm

Algorithm 1 ContraPrompt: Contrastive Prompt Optimization

Require: Training set $\mathcal{D}_{\text{train}}$, base module M, max iterations T, max attempts A, patience P
Ensure: Optimized module M^*

$\mathcal{P} \leftarrow$ ExtractInstructions(M); $\mathcal{R}_{\text{all}} \leftarrow \emptyset$; $s^* \leftarrow -1$; wait $\leftarrow 0$
for $t = 1$ to T **do**
  {**Phase 1: Instrumented agentic retry loop**}
  $\mathcal{A} \leftarrow \emptyset$ {attempt records across all training examples}
  for each $x \in \mathcal{D}_{\text{train}}$ **do**
    $\{(a_x^1, \tau_x^1), \ldots, (a_x^A, \tau_x^A)\} \leftarrow$ SolveWithRetries(M, x, A)
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(x, \{(a_x^j, \tau_x^j)\}_{j=1}^{A})\}$
  end for
  {**Phase 2: Contrastive pair mining (per-example)**}
  $\mathcal{C} \leftarrow$ MineContrastivePairs$(\mathcal{A})$ $\{\Delta_i \geq \delta_{\min}\}$
  {**Phase 3: Rule extraction and failure analysis**}
  $\mathcal{R}_{\text{contrastive}} \leftarrow \emptyset$
  for each pair $(\tau^-, \tau^+) \in \mathcal{C}$ **do**
    $r \leftarrow$ ExtractRule(τ^-, τ^+) {Dyadic trace analysis}
    $\mathcal{R}_{\text{contrastive}} \leftarrow \mathcal{R}_{\text{contrastive}} \cup \{r\}$
  end for
  $\mathcal{R}_{\text{failure}} \leftarrow$ AggregatedFailureAnalysis(all-fail examples in $\mathcal{A}$)
  $\mathcal{R}_{\text{all}} \leftarrow \mathcal{R}_{\text{all}} \cup \mathcal{R}_{\text{contrastive}} \cup \mathcal{R}_{\text{failure}}$
  {**Phase 4: Input-aware tree merge**}
  $\mathcal{T} \leftarrow$ TreeMerge$(\mathcal{R}_{\text{all}}$, failing inputs in $\mathcal{A})$
  {**Phase 5: Inject and checkpoint**}
  $M_t \leftarrow$ InjectTree$(M, \mathcal{T})$
  $s \leftarrow$ Evaluate$(\mathcal{D}_{\text{train}}, M_t)$
  if $s > s^*$ **then**
    $s^* \leftarrow s$; $M^* \leftarrow M_t$; wait $\leftarrow 0$
  else
    wait $\leftarrow$ wait $+1$
  end if
  if wait $\geq P$ **then**
    break
  end if
end for
return M^*

B Experimental Setup Details

Data splits. For HotPotQA, BBH, and GPQA Diamond, we use 50 training and 50 validation examples. Test evaluation uses 200 examples for HotPotQA, 540 for BBH, and 98 for GPQA Diamond. For GDPR-Bench, we use 100 training, 100 validation, and 687 test samples.

ContraPrompt configuration. A=3 maximum attempts, up to T=15 outer iterations with patience P=3, minimum improvement threshold $\delta_{\min}$=0.02, maximum two levels of tree nesting. No benchmark-specific tuning.

GEPA comparison protocol. GEPA's full HotPotQA pipeline uses a four-module architecture and achieves 62.33 F1 on Qwen3-8B; we disclose this for context but that system answers a different question. All methods (baseline, GEPA, ContraPrompt) operate under the same single-module constraint on the same model.

C Ablation Studies

Figure 3 presents component ablations, removing one component at a time from the full system and measuring the average relative performance drop across all four benchmarks. "Average relative drop" is computed as the mean, across the four benchmarks, of $(\text{score}_{\text{full}} - \text{score}_{\text{ablated}})/\text{score}_{\text{full}}$.

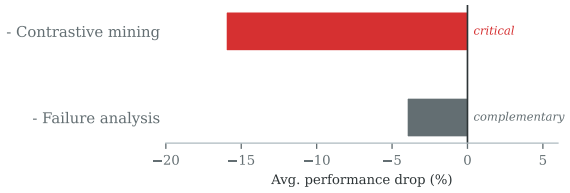

Figure 3: Average relative performance drop when each component is removed. Contrastive mining produces the largest degradation (-16% relative); the input-aware tree and aggregated failure analysis provide smaller complementary gains.

Removing multi-attempt solving and contrastive pair mining produces a -16% average relative drop. Without contrastive pairs, the method reduces to failure-only analysis, which provides a negative signal (what to avoid) but no positive target (what to do instead). The deficit is largest on HotPotQA (-13.8 percentage points of F1), precisely where retry success rate and contrastive pair volume are highest. This is consistent with the central claim that the dyadic trace comparison is the primary optimization signal.

Replacing the input-aware tree with flat rule injection produces a -6% average relative drop. The benefit is concentrated on benchmarks with heterogeneous input types (HotPotQA, GPQA Diamond), where irrelevant rules interfere with processing of out-of-scope inputs.

Removing aggregated failure analysis produces a -4% average relative drop, consistent with its role as a fallback for the capability-deficit regime where contrastive pairs are unavailable.

Trace-level vs. answer-only extraction. To directly test whether reasoning traces carry signal beyond the pairing structure, we ran a variant of the rule extractor that receives only final answers rather than complete chain-of-thought traces. This variant matches the performance of removing contrastive mining entirely (-16% average relative drop), indicating that the reasoning trace, not merely the paired comparison structure, is the source of the step-level optimization signal. This distinguishes ContraPrompt from final-output contrastive methods such as DPO and prompt-level contrastive approaches.

D Analysis: Interpretability of Extracted Rules

The rules extracted by dyadic trace analysis target specific reasoning steps by construction: each rule describes a reasoning step present in a successful trace and absent in a failed trace on the same input.

Representative rules from three benchmarks illustrate the specificity achievable:

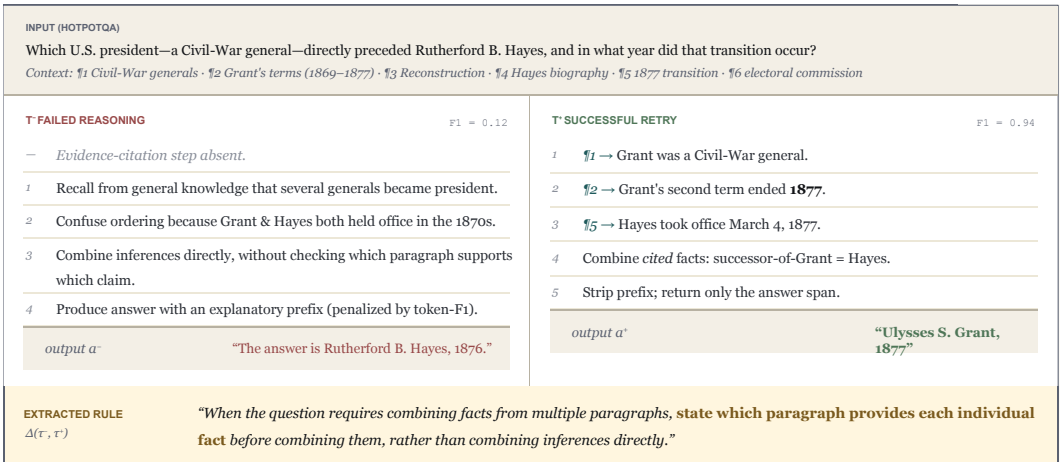

Figure 4: A representative contrastive pair on HotPotQA. Both traces share model, input, and base prompt; they differ in reasoning strategy (and, as a consequence of the retry mechanism, the appended error feedback). The extracted rule targets the specific reasoning step that was inserted on retry—explicit paragraph-level attribution before combining facts—rather than the generic prescription "be more thorough".

Example

HotPotQA: "When the question requires combining facts from multiple paragraphs, state which paragraph provides each individual fact before combining them, rather than combining inferences directly."

GPQA Diamond: "When a question involves a quantitative physical calculation, verify that the units and order of magnitude of the intermediate result are consistent with the physical scenario before proceeding to the final answer."

BBH (object tracking): "When the task involves tracking object states through a sequence of swap operations, maintain an explicit ordered state list that is updated after each individual operation rather than reasoning about the final state directly."

Each rule names a specific step in the reasoning chain rather than offering generic advice. This specificity follows from extracting rules from trace comparisons rather than from monadic failure diagnoses. GEPA's monadic reflection tends to produce higher-level prescriptions about thoroughness. Both rule types are useful; the two methods are complementary.

E Black-Box Optimization: Illustrative Examples

We highlight representative problems illustrating behavioral differences between ContraPrompt, GEPA `optimize_anything`, and Optuna. Complete artifacts are available in our repository here.

Easom (5D). The Easom function has a single narrow basin near $(\pi, \pi, \ldots, \pi)$ and is nearly flat across the rest of its domain. Optuna's 2,000 TPE trials never locate this basin (gap 5.03), and GEPA fares worse (gap 15.5), exhausting its budget on global exploration without concentrating search near the basin. Contrastive analysis of best versus worst evaluations produces rules identifying that improved solutions cluster near the center of the positive quadrant; from these, the LLM synthesizes a solver that enumerates candidate regions around $(\pi, \pi, \ldots, \pi)$, runs coordinate descent along each dimension independently, and refines with L-BFGS-B from diverse starting points. The final solver finds the exact global minimum (gap $< 10^{-15}$) in 194 lines of Python. No human specified this strategy; the contrastive landscape summary led the LLM to identify the basin location from evaluation pairs. This problem exemplifies why structured landscape summaries outperform raw trial logs: without contrastive pairs identifying the flat-versus-basin structure, the reflection LLM has no signal to motivate targeted search.

McCourt10 (8D). Both ContraPrompt and GEPA converge to the exact global minimum of -2.519 to 12 decimal places (gap $< 5 \times 10^{-12}$), while Optuna stalls at a gap of 0.015. Both synthesized solvers follow a similar three-phase pipeline: warm-start from prior best points, differential evolution with the best known solution seeded into the initial population, and L-BFGS-B local refinement. This is one of the problems where both code evolution approaches converge to effectively identical strategies, suggesting that when the landscape provides clear gradient signal, either feedback mechanism suffices.

McCourt11 (8D). The sole problem where GEPA outperforms ContraPrompt. GEPA reaches a gap of 3.9×10^{-8} from the global minimum ($f^* = -0.3905$), while ContraPrompt stalls at a gap of 0.074. GEPA's winning solver chains Gaussian process surrogate modeling with multi-start local optimization and dual annealing, allocating budget adaptively based on improvement rate. ContraPrompt's contrastive rules correctly identify the promising region but the synthesized code underallocates budget to local refinement, suggesting that on landscapes where surrogate-guided search is critical, GEPA's richer ASI (which includes all prior trial coordinates) can outperform contrastive summaries that compress this information into textual rules.

Ackley (11D). Contrastive rules surface that the function is approximately separable, leading ContraPrompt to synthesize a six-phase pipeline leading with coordinate-wise search before chaining Powell's method, Nelder-Mead, and L-BFGS-B. The final gap of 3.26 improves on Optuna's 5.52 by 41%. GEPA produces a gap of 15.1, nearly $5\times$ worse, despite generating a longer solver. Without contrastive rules identifying separability, the solver allocates most budget to full-dimensional differential evolution, which is inefficient in 11 dimensions at the 2,000-evaluation scale.

DeflectedCorrugatedSpring (4D). ContraPrompt finds the exact global minimum ($f^* = -1.0$, gap $< 10^{-13}$), while both GEPA and Optuna converge to a local minimum at $f = -0.843$. The chronology is as follows: early random search within the code evolution loop encounters an evaluation near -1.0 by chance; this evaluation is recorded alongside the dominant -0.843 local attractor evaluations. The contrastive pair between -0.843 and -1.0 exposes the existence of the deeper basin, prompting the LLM to synthesize a solver with explicit multi-basin detection: it restarts from random initializations after each local convergence, maintaining a diverse archive of local optima. The subsequent generalized solver reliably relocates and refines the deep basin across restarts. Neither GEPA's reflection nor Optuna's TPE sampler construct this multi-basin logic, and both remain trapped at -0.843.