EEVEE: Towards Test-time Prompt Learning in the Real World for Self-Improving Agents

Weixian Xu* **Shilong Liu†** **Mengdi Wang†**
Princeton University

Abstract

In this paper, we propose EEVEE, the first *multi-dataset test-time prompt learning* framework for LLM agents, enabling test-time prompt learning under real-world task streams. Existing methods are largely designed for single-dataset settings, which limits their practical applicability: real-world deployments require models to handle heterogeneous input streams drawn from multiple datasets, domains, and task distributions. To mitigate cross-dataset interference, EEVEE introduces a *router* that partitions incoming inputs into task clusters and assigns them to suitable prompt configurations. This design is optimized via a *router-prompt co-evolution strategy*, which employs interleaved router and prompt learning phases to address their mutual dependency. Experiments across multiple datasets demonstrate that the framework improves robustness under heterogeneous data streams while maintaining single-benchmark learning capability and efficiency. Specifically, EEVEE improves average multi-benchmark scores by 10.38 and 24.32 points over Qwen3-4B-Instruct and DeepSeek-V3.2, surpassing SOTA methods GEPA and ACE by up to 37.2% and 48.2%.

1 Introduction

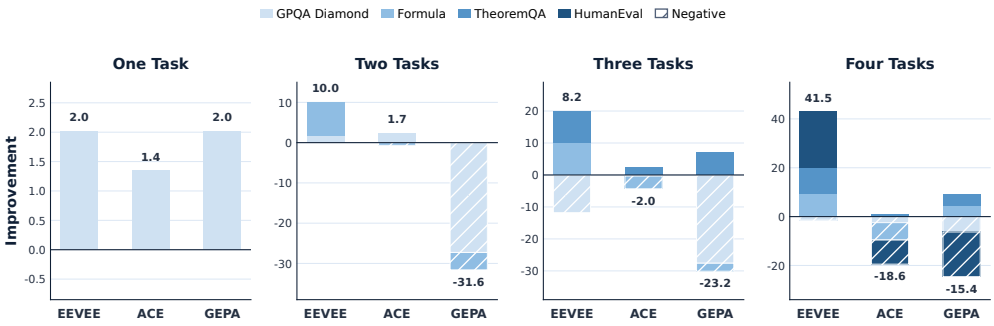


Figure 1: Incremental multi-benchmark retention as tasks are added in order: GPQA Diamond, Formula, TheoremQA, and HumanEval. Each bar stacks per-benchmark improvements for tasks seen so far: solid upward blocks are gains, and hatched downward blocks are retention losses. The number above or below each bar is the summed improvement.

Test-time prompt learning offers a lightweight mechanism for adapting foundation models after deployment. Prior work shows that prompts can serve as an effective adaptation interface without updating model weights Lester et al. (2021), and black-box optimizers can revise instructions from model feedback Pryzant et al. (2023); Yang et al. (2024). Rather than relying on a fixed offline prompt, test-time prompt learning updates prompts for new inputs, distribution shifts, and failure modes. This makes it suitable for self-improving agents, where behavior is refined through interaction with the environment, as in reflective agents and evolving-context systems Shinn et al. (2023); Zhang et al. (2026).

* Work done during a remote internship at Princeton University. †Corresponding authors.

Recent methods such as GEPA Agrawal et al. (2026), ACE Zhang et al. (2026), and Combee Li et al. (2026) improve test-time prompt learning through reflection, context evolution, or scalable trace aggregation. However, they mostly adapt within a single dataset or benchmark. In real-world deployment, incoming queries often come from heterogeneous domains, task formats, and capability mixtures. We formalize this regime as *multi-dataset test-time prompt learning*: a model receives a stream of examples drawn from multiple datasets and domains rather than one stationary source.

This setting exposes cross-dataset interference. Existing methods often assume a unified adaptation objective, a fixed prompt space, or feedback from one benchmark, so updates for one domain can harm another. As Figure 1 shows, when more benchmarks enter the adaptation stream, GEPA Agrawal et al. (2026) and ACE Zhang et al. (2026) accumulate negative retention on previous tasks, suggesting that a single learned prompt struggles to absorb heterogeneous feedback without losing task-specific behavior. This motivates a framework that can preserve specialization while still learning from a mixed stream of tasks.

We propose EEVEE, a test-time prompt learning framework that augments prompt learning with a *router*. Instead of forcing all inputs through one adaptation path, the router partitions the stream into task clusters and assigns each cluster to a suitable prompt configuration. This preserves prompt-based adaptation while reducing destructive interference across domains.

Designing the router, however, is itself difficult. A rigid router fails to capture diverse task structure, while an unstable router disrupts prompt optimization. The prompt learner and router are also coupled: routing determines which examples each prompt learns from, and prompt behavior determines which routing policy is useful. We therefore introduce a *router-prompt co-evolution strategy* that interleaves router and prompt learning phases, allowing routing decisions and prompt updates to improve together rather than being fixed or trained in isolation. To make this co-evolution practical, we further design a three-stage training process that initializes useful prompt slots, explores coupled updates efficiently, and then converges under a stable router.

We evaluate EEVEE on multiple datasets and show that it consistently outperforms competitive baselines for multi-dataset test-time prompt learning. Across the four-benchmark suite, EEVEE improves the average score by 10.38 and 24.32 points over Qwen3-4B-Instruct Yang et al. (2025) and DeepSeek-V3.2 DeepSeek-AI et al. (2025), and by up to 37.2% and 48.2% over GEPA and ACE. In the incremental multi-benchmark setting, EEVEE ends with a +41.53 cumulative retention gain after all tasks are introduced, while GEPA and ACE end at -15.36 and -18.58. Single-benchmark and token-cost analyses further show that the routing design remains competitive in conventional single-task settings while avoiding ACE's large prompt expansion.

Contributions:

- We propose EEVEE, the first *multi-dataset test-time prompt learning* framework for LLM agents, using a router to reduce cross-dataset interference.
- We introduce *router-prompt co-evolution*, enabled by a three-stage training design, to jointly improve router prompts and model prompts through interleaved phases.
- We validate EEVEE on multiple datasets, showing strong performance, retention, and efficiency, with case studies that offer practical guidance.

2 Methods

2.1 Framework Overview

EEVEE targets multi-dataset test-time prompt learning, where a mixed stream contains different domains, formats, and evaluation rules. A single evolving prompt can let updates for one task family interfere with another, so EEVEE maintains a set of specialized prompts

and a router that chooses which prompt should handle each input. We denote the prompt set by $\mathcal{P} = \{p_1, \ldots, p_K\}$ and the router by R. The target model M is fixed.

At inference time, the router first selects a slot and the target model then answers with the corresponding prompt:

$$z = R(x; \mathcal{P}) \in \{1, \ldots, K\}, \qquad \hat{y} = M(x; p_z).$$

This preserves prompt-based adaptation while allowing different inputs to invoke different specialized behaviors.

To obtain strong performance, the router itself must be learned: default or manually written routers do not reliably recover behaviorally useful partitions, as shown in Table 2. Router learning iterates the router prompt and has a mutual dependency with prompt learning: routing determines which examples each prompt sees, while prompt quality determines whether a routing decision is good. EEVEE therefore learns both through router-prompt co-evolution.

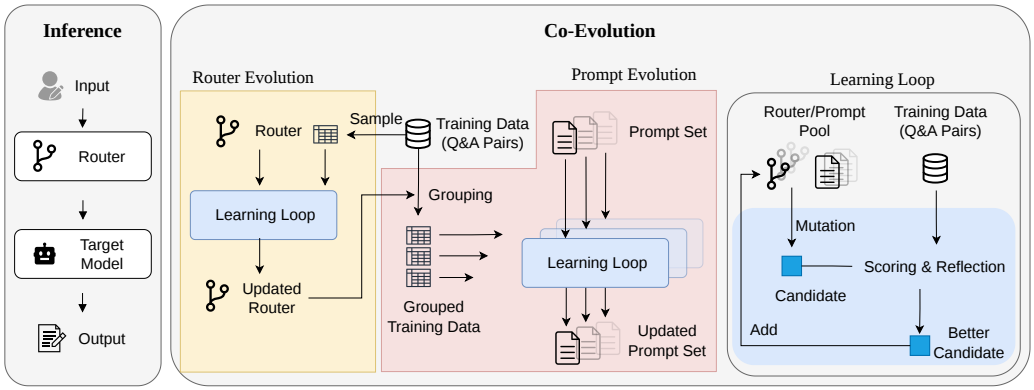


Figure 2: Main framework of EEVEE. Inference routes each input to a specialized prompt; learning co-evolves the router and prompt set through mutation, analysis, reflection, scoring, and regrouping.

2.2 Router-Prompt Co-evolution

The mixed adaptation data is split into training and validation sets, $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$. Each co-evolution cycle alternates two operations: router evolution fixes the prompt set and searches for a better router, then prompt evolution fixes this router and updates each slot prompt on its routed data. Write $\mathcal{P}_T = \{p_{k,T}\}_{k=1}^{K}$. In a cycle starting from R_T and $\mathcal{P}_T$,

$$R_{T+1} = \text{RouterEvolve}(R_T; \mathcal{P}_T, \mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}}), \qquad \mathcal{P}_{T+1} = \mathcal{P}_T,$$
$$\mathcal{P}_{T+2} = \left\{\text{PromptEvolve}(p_{k,T+1}; R_{T+1}, \mathcal{D}_{\text{tr},k}^{T+1}, \mathcal{D}_{\text{val},k}^{T+1})\right\}_{k=1}^{K}, \quad R_{T+2} = R_{T+1}.$$

The next router phase uses the updated prompt set, so routing decisions and prompt specialization improve each other rather than being optimized in isolation.

Router evolve. Given R_T and fixed prompts $\mathcal{P}_T$, EEVEE initializes a temporary router pool $\mathcal{B}_R$ and repeatedly samples a router mini-batch $\mathcal{D}_{\text{RM}}$. To make errors attributable to assignment rather than prompt incapability, $\mathcal{D}_{\text{RM}}$ is sampled only from training examples that at least one current slot prompt can solve. Each update step samples reference routers from $\mathcal{B}_R$ and mutates them into R_{mut}, allowing the search to redesign routing rules rather than only refine an existing router.

EEVEE evaluates R_{mut} on $\mathcal{D}_{\text{RM}}$. Since router quality is observed through downstream prompt correctness, EEVEE analyzes cases where the routed slot fails but another slot succeeds, explaining why the better slot matches the task. Reflection uses these analyses

and ground-truth correctness to produce R_{ref}. Let $s_{\text{mb}}(\cdot)$ denote mini-batch downstream score. EEVEE keeps

$$R^\star = \arg\max_{R \in \{R_{\text{mut}}, R_{\text{ref}}\}} s_{\text{mb}}(R),$$

evaluates $R^\star$ on $\mathcal{D}_{\text{val}}$, and admits it into $\mathcal{B}_R$ only if it improves over the phase baseline R_T. When the router score plateaus, the phase outputs R_{T+1} while keeping $\mathcal{P}_T$ fixed.

Router candidates are scored by downstream accuracy, consistency, and balance. Let $a_R(x,y) = \mathbf{1}[M(x; p_{R(x;\mathcal{P}_T)}) = y]$ denote routed correctness under R and the fixed prompt set:

$$\begin{aligned} S_R(R) &= \lambda_{\text{acc}} A(R) + \lambda_{\text{con}} C(R) + \lambda_{\text{bal}} B(R), & A(R) &= \tfrac{1}{|\mathcal{D}_{\text{val}}|} \textstyle\sum_{(x,y)\in\mathcal{D}_{\text{val}}} a_R(x,y), \\ C(R) &= \beta_{\text{in}}\, \text{Compact}(R) + \beta_{\text{out}}\, \text{Separate}(R), & B(R) &= \gamma_{\text{use}} \tfrac{|\mathcal{K}_R|}{K} + \gamma_{\text{bal}}\, \text{Balance}(\pi_R). \end{aligned}$$

Here $\mathcal{K}_R$ is the set of labels used by R and π_R is its empirical label distribution. Compact rewards same-label examples with similar cached correctness vectors, while Separate rewards behaviorally distinguishable labels. The weights are annealed from consistency and balance toward downstream accuracy. Appendix C.1 shows that performance is stable across alternative annealing, weighting, and prompt-search settings.

Prompt evolve. After router evolution, R_{T+1} routes the full data into slot-specific groups:

$$\mathcal{D}^{T+1}_{\text{tr},k} = \{(x,y) \in \mathcal{D}_{\text{tr}} : R_{T+1}(x;\mathcal{P}_T) = k\}, \quad \mathcal{D}^{T+1}_{\text{val},k} = \{(x,y) \in \mathcal{D}_{\text{val}} : R_{T+1}(x;\mathcal{P}_T) = k\}.$$

Each non-empty slot is evolved independently and in parallel. For slot k, EEVEE initializes a temporary prompt pool $\mathcal{B}_{P,k}$ and samples $\mathcal{D}_{\text{PM}} \subset \mathcal{D}^{T+1}_{\text{tr},k}$. Prompt evolution also uses mutation and reflection, but without the router-specific analysis step: it proposes p_{mut} from reference prompts and mini-batch examples, then directly reflects from question, target answer, model answer, and correctness to obtain p_{ref}. The better mini-batch prompt

$$p^\star = \arg\max_{p \in \{p_{\text{mut}}, p_{\text{ref}}\}} s_{\text{mb}}(p)$$

is evaluated on the routed validation set with score $s^k_{\text{val}}(\cdot)$.

EEVEE stores prompts in a Pareto-front pool Mouret & Clune (2015). Each prompt is represented by its correctness vector over $\mathcal{D}^{T+1}_{\text{val},k}$; a prompt is dominated if another is at least as correct on every example and strictly better on one. Thus the frontier preserves complementary prompts. A candidate enters $\mathcal{B}_{P,k}$ only when

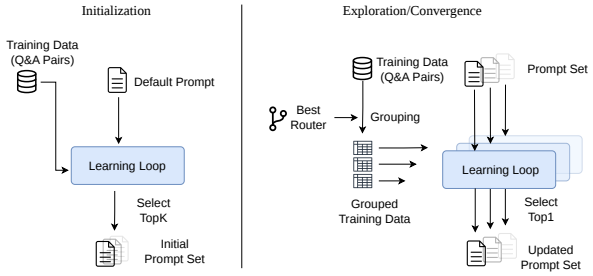
$$s^k_{\text{val}}(p^\star) > s^k_{\text{val}}(p_\emptyset), \qquad p^\star \in \text{ParetoFront}(\mathcal{B}_{P,k} \cup \{p^\star\}),$$

where $p_\emptyset$ is the empty prompt. The empty-prompt floor removes ineffective edits, and the Pareto-front rule preserves diverse useful prompts. When all non-empty slots plateau, prompt evolution returns the updated prompt set.

2.3 Training Stages

EEVEE uses three stages with distinct roles: initialization creates usable prompt slots, exploration searches over coupled router-prompt designs, and convergence spends a larger budget after routing stabilizes.

Initialization. Router evolution needs a prompt set that can reveal whether a routing decision is useful; otherwise accuracy reflects prompt weakness rather than router quality. EEVEE therefore first initializes a diverse prompt set before router learning. Initialization runs prompt learning on the mixed training set and keeps a Pareto-front pool: because frontier prompts cover complementary examples, selecting from this pool yields specialized prompts with distinguishable behavior. Let $\mathcal{B}$ be the initialization pool after dominated candidates are removed, and let F_i be the reduced Pareto frontier for validation example i. With coverage $C_F(p) = \{i : p \in F_i\}$ and $\mathcal{S}_0 = \emptyset$, EEVEE greedily retains prompts by the rule in Figure 3, where Δ denotes additional coverage. The retained prompts form $\mathcal{P}^0$ and provide specialized behaviors from which the initial router can be written.



$$p_k = \arg\max_{p \in \mathcal{B} \setminus \mathcal{S}_{k-1}} \Delta(p; \mathcal{S}_{k-1}),$$
$$\Delta(p; \mathcal{S}) = \left| C_F(p) \setminus \bigcup_{q \in \mathcal{S}} C_F(q) \right|,$$
$$\mathcal{S}_k = \mathcal{S}_{k-1} \cup \{p_k\}.$$

Figure 3: Three-stage training design and initialization rule. Left: initialization builds prompts, exploration alternates router and prompt evolution, and convergence fixes the router for larger-budget prompt learning. Right: greedy coverage rule for top-K prompt selection.

Exploration. Exploration starts from $(R_0, \mathcal{P}^0)$ and alternates router and prompt evolution under lightweight budgets. Frequent switching is necessary for efficiency: fully optimizing prompts under an unstable router wastes budget, while optimizing a router against stale prompts can overfit to obsolete prompt behavior. As described in Section 2.2, EEVEE enables an annealing mechanism for router-candidate scoring, shifting weights from consistency and balance toward downstream accuracy so early steps preserve diverse routing behaviors and later steps favor accurate, stable routing.

Convergence. Because exploration must switch phases frequently, it cannot fully optimize every slot prompt. Once annealing and alternating updates identify a stable router, convergence fixes $R^\star$, reroutes $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$, and spends a larger prompt-learning budget within each slot. This enables EEVEE to find strong prompts under a fixed router rather than continuing to move the partition.

3 Experiments

3.1 Settings

We evaluate EEVEE in a *multi-dataset test-time prompt learning* setting over four benchmarks: GPQA Diamond Rein et al. (2023), Formula Wang et al. (2025), TheoremQA Chen et al. (2023), and HumanEval Chen et al. (2021). GPQA Diamond tests closed-book knowledge QA; Formula and TheoremQA emphasize mathematical and symbolic reasoning; HumanEval evaluates code generation. Unless otherwise stated, methods learn from the mixed training stream and are scored on held-out test examples outside that stream. To reduce randomness, we run stochastic settings multiple times and report averaged scores.

3.2 Main Results

Table 1 compares average performance over three runs against two strong reflection-based prompt-learning baselines, GEPA Agrawal et al. (2026) and ACE Zhang et al. (2026), plus the unadapted target model. On Qwen3-4B-Instruct Yang et al. (2025), EEVEE reaches 51.75 average score, improving over the target-model baseline by 10.38 points and outperforming GEPA and ACE by 14.02 and 16.83 points. Its gains over the baseline are +9.33 on Formula, +10.48 on TheoremQA, and +23.17 on HumanEval, where the final score reaches 72.63. On DeepSeek-V3.2 DeepSeek-AI et al. (2025), EEVEE reaches 64.07 average score, improving over the target-model baseline by 24.32 points and over GEPA by 8.24 points; the per-benchmark gains are +30.55 on Formula, +18.63 on TheoremQA, and +50.00 on HumanEval, with HumanEval reaching 92.82.

3.3 Ablations

Table 2 isolates the main components on Qwen3-4B-Instruct. Besides the unadapted baseline, we test a default router that replaces the learned routing field with a blank or simple default

Table 1: Main results on the four-benchmark suite. Scores are percentages averaged over three runs; colored subscripts denote differences from the target-model baseline.

Target model	Method	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
Qwen3-4B Instruct	Baseline	56.00	45.22	14.79	49.46	41.37
	ACE	48.93$_{-7.07}$	39.67$_{-5.55}$	15.84$_{+1.05}$	35.23$_{-14.23}$	34.92$_{-6.45}$
	GEPA	50.84$_{-5.16}$	49.83$_{+4.61}$	19.62$_{+4.83}$	30.62$_{-18.84}$	37.73$_{-3.64}$
	EEVEE	54.55$_{-1.45}$	54.55$_{+9.33}$	25.27$_{+10.48}$	72.63$_{+23.17}$	**51.75**$_{+10.38}$
DeepSeek-V3.2	Baseline	64.98	30.00	21.21	42.82	39.75
	ACE	55.89$_{-9.09}$	37.78$_{+7.78}$	27.05$_{+5.84}$	78.59$_{+35.77}$	49.83$_{+10.08}$
	GEPA	41.75$_{-23.23}$	60.56$_{+30.56}$	31.72$_{+10.51}$	89.29$_{+46.47}$	55.83$_{+16.08}$
	EEVEE	63.08$_{-1.90}$	60.55$_{+30.55}$	39.84$_{+18.63}$	92.82$_{+50.00}$	**64.07**$_{+24.32}$

field, a manual router written once by GPT-5.4 and then fixed, and a no-co-evolution variant that first learns the router and then learns prompts in a separate second stage.

The full method reaches 51.75 average score, 8.17 points above the default router (43.58), 14.57 above the manual router (37.18), and 8.87 above no co-evolution (42.88). Default routing and no co-evolution improve the baseline by only 2.21 and 1.51 points, respectively, while the manual router is 4.19 points below the baseline. These gaps show that EEVEE needs both learned routing and interleaved router–prompt optimization; static partitions or two-stage learning do not capture the mutual dependence between routing decisions and prompt behavior. Appendices B.1, B.4, and B.7 add equal-budget random, classifier, and oracle dataset-ID routers, none of which reach EEVEE, the partition the router learns, a slot-count sweep, and the same ablations on DeepSeek-V3.2.

Table 2: Ablation results for the main components of EEVEE on Qwen3-4B-Instruct.

Variant	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
Baseline	56.00	45.22	14.79	49.46	41.37
Default Router	47.03	47.11	26.10	54.06	43.58
Manual Router	35.24	42.11	23.96	47.42	37.18
No Co-evolution	39.84	47.39	24.81	59.49	42.88
EEVEE	54.55	54.55	25.27	72.63	51.75

3.4 Task Scaling

We first check whether EEVEE remains competitive when prompt learning is run on one benchmark, then study what happens as the number of jointly learned benchmarks increases. In the single-benchmark setting in Figure 4, EEVEE is broadly competitive with GEPA and ACE. Figure 4 also includes FiNER Loukas et al. (2022) and IFBench Pyatkin et al. (2025) to align with benchmarks tested by prior methods. It reaches 55.25 on Formula and 73.17 on HumanEval, outperforming both baselines on these two benchmarks, and improves TheoremQA to 25.27. The no-router variant also performs strongly on Formula and HumanEval, showing that our prompt learning design itself is effective even without router specialization.

The difference becomes clearer as the benchmark mixture grows. Figure 1 shows that GEPA and ACE quickly lose retention as more tasks are added; after all four tasks, both end below zero. In contrast, EEVEE remains positive throughout and ends at +41.53 cumulative retention. Thus the main advantage appears when the task mixture grows: a single learned prompt loses retention, while router-conditioned prompt learning preserves positive net improvement. Appendix B.6 varies the mixture composition and task order.

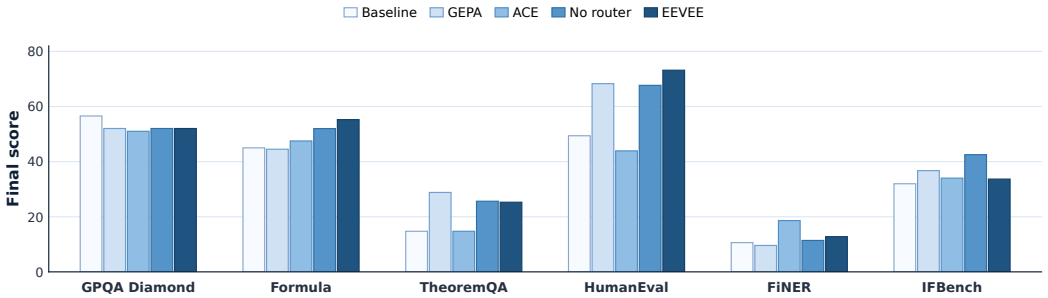


Figure 4: Single-benchmark results Loukas et al. (2022); Pyatkin et al. (2025). Bars show final scores after learning independently on each benchmark. The no-router variant keeps one retained prompt, isolating prompt learning without router specialization.

3.5 Generalization

We test two forms of generalization: cross-model generalization, which asks whether prompts learned on one target model still help a different target model, and cross-task generalization, which asks whether prompts learned on the four primary benchmarks generalize to unseen tasks. For cross-model generalization, prompts learned with Qwen3-4B-Instruct Yang et al. (2025) are applied directly to DeepSeek-V3.2 DeepSeek-AI et al. (2025) in non-thinking mode. For cross-task generalization, prompts learned on the four primary benchmarks are evaluated on held-out MBPP Austin et al. (2021) and MMLU-Pro Wang et al. (2024). MBPP is close to HumanEval and tests coding-domain generalization; MMLU-Pro is broader knowledge QA and tests unrelated-domain robustness.

Table 3: Cross-model and held-out benchmark generalization. Left: prompts learned with Qwen3-4B-Instruct are evaluated on DeepSeek-V3.2 alongside the source-model result. Right: prompts learned on the main four-benchmark suite are evaluated on MBPP and MMLU-Pro with a shared unadapted baseline. Scores are averaged over three runs.

Cross-Model Generalization

Setting	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
DeepSeek base	64.98	30.00	21.21	42.82	39.75
Qwen3 EEVEE	54.55	54.55	25.27	72.63	51.75
Qwen3 transfer	60.27	42.28	32.89	77.04	**54.10**

Held-Out Generalization

Method	MBPP	MMLU Pro	Avg.
Baseline	69.29	70.74	70.01
EEVEE	**70.42**	68.92	69.67
GEPA	68.20	68.85	68.53
ACE	67.47	69.32	68.39

Table 3 summarizes both settings. In cross-model generalization, the prompts learned on Qwen3-4B-Instruct raise the DeepSeek-V3.2 average from 39.75 to 54.10, with gains of +12.28 on Formula, +11.68 on TheoremQA, and +34.22 on HumanEval. In cross-task generalization, EEVEE improves MBPP from 69.29 to 70.42, while GEPA and ACE drop to 68.20 and 67.47. On MMLU-Pro, EEVEE decreases from 70.74 to 68.92, a 1.82-point drop that is smaller than GEPA's 1.89-point drop and comparable to ACE's 1.42-point drop. This pattern is aligned with our later case study, where we further analyze its underlying causes. Appendix B.8 recovers most of the MMLU-Pro loss with an inference-time gate.

3.6 Token Cost

EEVEE adds a router before answer generation, so we measure final-test token cost. Averaged over the four benchmarks in Figure 5, EEVEE uses 4.32k total tokens per example, close to GEPA's 3.47k and far below ACE's 21.30k. The input-token gap is similar: EEVEE uses 3.00k input tokens on average, compared with 2.44k for GEPA and 20.35k for ACE. ACE incrementally edits playbook bullets, which can accumulate redundant entries and lengthen prompts as tasks and data grow. EEVEE's router therefore adds only modest overhead while using about 4.9× fewer total tokens than ACE. Appendices B.3 and B.9 add the adaptation-phase cost and its scaling.

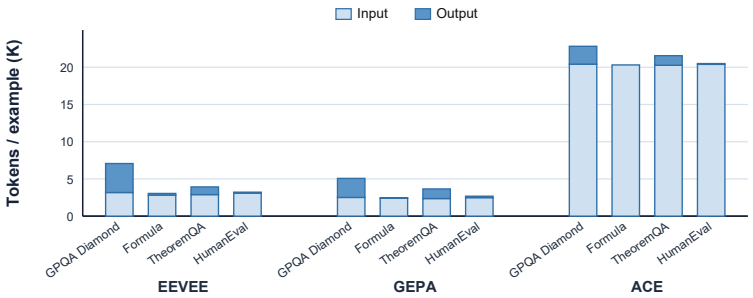


Figure 5: Average token usage per test example with Qwen3-4B-Instruct. Each method group contains four benchmark bars, each stacked into input and output tokens.

3.7 Case Study: What Does Prompt Learning Capture?

We retest six completed EEVEE runs by comparing the empty prompt with the final learned router and prompt set on the same held-out examples: three runs with Qwen3-4B-Instruct and three runs with DeepSeek-V3.2. Table 4 reports the flip summary; the cases below show what these flips reveal, with representative learned prompt and output excerpts in Appendix A.

Table 4: Six-run empty-vs-final retest over three Qwen3-4B-Instruct and three DeepSeek-V3.2 runs. Δ is average final-minus-empty change; W→R and R→W count per-example flips in that order; Runs+ counts positive-Δ runs.

Type	Benchmark	Qwen Δ	DeepSeek Δ	W→R; R→W	Runs+
Code	HumanEval	+23.2	+48.8	193; 16	6 of 6
Formula	Formula	+9.5	+31.7	268; 21	6 of 6
Math QA	TheoremQA	+10.0	+18.2	632; 166	6 of 6
Domain QA	GPQA Diamond	-3.7	-7.7	55; 89	1 of 6
Overall	Mixed suite	+9.6	+21.1	1148; 292	6 of 6

The table suggests a task-property pattern rather than a benchmark-specific one. Gains are larger on code and formula tasks, where feedback can be turned into reusable rules for how to solve and how to present the answer. TheoremQA also improves, but with more mixed flips because it combines computation, symbolic reasoning, and answer parsing. GPQA Diamond is the only benchmark with more regressions than recoveries, suggesting that stronger learned reasoning can sometimes underweight the domain knowledge needed for closed-book QA. Figure 6 shows two positive flips and one representative negative flip.

Formula: unit scale
Task: Compute free cash flow from operating cash flow and capital expenditure.
Baseline: Reverses subtraction and keeps the million-scale decimal, producing a negative value.
Learned: Applies the dollar-scale formula and emits 400000.00.

HumanEval: executable body
Task: Sum even values at odd indices.
Baseline: Writes a bare expression without the required indented return.
Learned: Produces an executable body with an accumulator and return.

GPQA Diamond: knowledge underweighted
Task: Select the densest Earth-like exoplanet from mass and composition cues.
Baseline: Uses the rocky-planet prior that higher mass increases self-compression and density.
Learned: Uses formulaic density reasoning, treats same composition as constant density, and selects Earth.

Figure 6: Case-study flips. Boxes summarize behavior; excerpts are in Appendix A.

These cases show why prompt learning can improve and hurt. For executable programs and formula-grounded computation, feedback teaches reusable procedures: preserve interfaces, edge cases, units, and parseable answers. For knowledge-intensive QA, learned prompts

strengthen generic reasoning and answer comparison rather than add missing knowledge. In GPQA Diamond, the learned response follows density formulas but assumes constant density, underweighting gravitational compression of rocky planets.

Takeaway. Prompt learning favors reusable procedures but may underweight domain knowledge.

4 Related Work

Prompt learning. Prompt learning began with soft prompts, prefixes, or discrete triggers Shin et al. (2020); Li & Liang (2021); Lester et al. (2021); later black-box and population-based methods optimized prompts or programs from scores or feedback Zhou et al. (2022); Yang et al. (2024); Pryzant et al. (2023); Guo et al. (2024); Fernando et al. (2023); Khattab et al. (2023); Opsahl-Ong et al. (2024); Yuksekgonul et al. (2024). Most closely, GEPA Agrawal et al. (2026) combines natural-language reflection and Pareto-front selection Mouret & Clune (2015), ACE maintains an adaptive playbook Zhang et al. (2026), and Combee aggregates parallel traces Li et al. (2026). They improve adaptation but target one distribution or shared prompt context. Memory-extraction work learns cross-dataset prompts, not general LLM-agent capabilities Yang et al. (2026). EEVEE learns router-conditioned prompts for task-general mixed-dataset streams.

Self-improving agents. Self-improving agents use feedback loops: Self-Refine and Reflexion use natural-language feedback or verbal memory Madaan et al. (2023); Shinn et al. (2023), while generative agents and Voyager maintain memories or skill libraries Park et al. (2023); Wang et al. (2023). Evolutionary discovery agents use related loops for scientific and algorithmic search, including code evolution Novikov et al. (2025); Sharma (2025); Liu et al. (2026b); Cemri et al. (2026); Liu et al. (2026a); Xu et al. (2026). Together, these systems show histories can drive improvement in scoped programs or algorithms; EEVEE targets heterogeneous tasks, decoupling multi-task commonality from task-specific behavior.

Table 5: Comparison on adaptation, routing, and co-evolution.

Method	Multi-dataset	Router	Co-evolution
Prompt tuning and search	×	×	×
GEPA, ACE, Combee	×	×	×
Reflexive agents	×	×	×
Evolutionary discovery agents	×	×	✓
EEVEE	✓	✓	✓

5 Conclusion

We introduced EEVEE, a multi-dataset test-time prompt learning framework for LLM agents facing heterogeneous task streams rather than one benchmark distribution. To reduce cross-dataset interference, EEVEE maintains router-conditioned prompts, so inputs can be assigned to prompts specialized for compatible task behavior. Because the router and prompts depend on each other, EEVEE uses a three-stage router-prompt co-evolution procedure to initialize useful prompts, explore coupled updates, and refine prompts under a stable router. This makes the framework better aligned with heterogeneous real-world agent workloads. Experiments show strong gains over prompt-learning baselines in mixed-dataset settings, with benefits increasing across tasks and reasonable held-out and cross-model transfer. Case studies suggest prompt learning works best when feedback becomes reusable procedures, output contracts, or task-solving strategies. Overall, EEVEE offers a practical self-improving method and lens for real-world test-time prompt learning.

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.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alexandros G. Dimakis, and Ion Stoica. AdaEvolve: Adaptive llm driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. *arXiv preprint arXiv:2305.12524*, 2023.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, et al. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

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

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

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of EMNLP*, 2021.

Hanchen Li, Runyuan He, Qizheng Zhang, Changxiu Ji, Qiuyang Mang, Xiaokun Chen, Lakshya A. Agrawal, Wei-Liang Liao, Eric Yang, Alvin Cheung, James Zou, Kunle Olukotun, Ion Stoica, and Joseph E. Gonzalez. Combee: Scaling prompt learning for self-improving language model agents. *arXiv preprint arXiv:2604.04247*, 2026.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL-IJCNLP*, 2021.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. EvoX: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026a.

Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Z. Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Alexandros G. Dimakis, Matei Zaharia, and Ion Stoica. SkyDiscover: A flexible framework for ai-driven scientific and algorithmic discovery, 2026b. URL https://skydiscover-ai.github.io/blog.html.

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 the 60th Annual Meeting of the Association for Computational Linguistics*, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Alexander Novikov, Ngan Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of EMNLP*, 2024.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of UIST*, 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 EMNLP*, 2023.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *Advances in Neural Information Processing Systems*, 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.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Taylor Shin, Yasaman Razeghi, Robert L. Logan, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of EMNLP*, 2020.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Dannong Wang, Jaisal Patel, Daochen Zha, Steve Y. Yang, and Xiao-Yang Liu. FinLoRA: Benchmarking LoRA methods for fine-tuning LLMs on financial datasets. *arXiv preprint arXiv:2505.19819*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Weixian Xu, Tiantian Mi, Yixiu Liu, Yang Nan, Zhimeng Zhou, Lyumanshan Ye, Lin Zhang, Yu Qiao, and Pengfei Liu. ASI-Evolve: Ai accelerates ai. *arXiv preprint arXiv:2603.29640*, 2026.

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

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

Yuqing Yang, Tengxiao Liu, Wang Bill Zhu, Taiwei Shi, Linxin Song, and Robin Jia. Self-evolving llm memory extraction across heterogeneous tasks. *arXiv preprint arXiv:2604.11610*, 2026.

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. In *International Conference on Learning Representations*, 2026.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Limitations and Social Impact

Although EEVEE improves multi-dataset prompt learning, limitations remain. Like other LLM-based evolutionary procedures, it cannot guarantee exact performance reproduction across runs, since stochastic search can produce different routers and prompt sets. Its feedback loop still relies on ground-truth or rule-based labels to accumulate task knowledge, so it is not yet a fully reflection-only learner; Appendix B.5 shows it degrades gracefully when that signal is sparse or noisy, but a deployment with no evaluative signal at all cannot drive updates. It does not require a prepared adaptation set: Appendix B.2 evaluates a strict test-then-adapt stream, although the resulting scores stay below the prepared-split results. A practical risk is distribution mismatch: if adaptation data is noisy or misaligned with the real application, learned prompts may generalize weakly or even degrade performance. Deployment should check data quality and distribution similarity before updates.

A Case Study Details

This appendix provides excerpts from the diagnostic retest discussed in Section 3.7. The retest compares the empty prompt against the final router and prompt set from six completed EEVEE runs: three Qwen3-4B-Instruct runs and three DeepSeek-V3.2 runs. The raw run logs store the full router decisions, answer calls, and per-example flip bundles. We do not reproduce all six prompt sets; instead, we include representative learned prompt excerpts from one Qwen3-4B-Instruct run and one DeepSeek-V3.2 run, followed by output excerpts for the three main-paper cases.

A.1 Learned Prompt Excerpts

Qwen3-4B code slot. In a representative Qwen3-4B-Instruct run, the code-oriented slot learned a task-execution policy for incomplete Python functions: preserve the interface, infer the exact continuation, cover edge cases, and avoid extra text that would break execution.

```
You are given a task description that defines an incomplete Python
function or problem to solve. Carefully read the input, extract
the exact task requirements, and generate only the precise output
```

```
expected by the task without extra text, explanations, markdown,
comments, or formatting.

Identify the task type:
- Writing a Python function body with specific inputs and outputs
- Solving a mathematical or logical problem
- Validating conditions such as bracket matching or palindrome checks
- Processing structured data such as strings, arrays, and dictionaries

Parse the input format and constraints:
- Note data types, edge cases, boundary values, and examples
- Pay close attention to ordering and special constraints

Understand the output contract:
- If the task says output ONLY the continuation,
  write only the function body.
```

Qwen3-4B science slot. The same representative Qwen3-4B-Instruct run routes many GPQA Diamond examples to a scientific-reasoning slot. This prompt encourages systematic physical and mathematical modeling, but it does not provide task-specific missing knowledge for every possible GPQA Diamond domain.

```
You are an expert problem solver specializing in analytical reasoning,
physical modeling, combinatorial mathematics, and scientific computation.
Your task is to answer precise, domain-specific questions that require
careful application of physical laws, mathematical principles, or
combinatorial logic, based on exact input constraints and known scientific
or mathematical facts.

For multiple-choice questions with options (A-D), select the single correct
choice and present it in the format: "Answer: (X)".

For physical or astronomical problems involving equilibrium, energy,
temperature, or orbital dynamics, apply known physical laws such as the
Stefan-Boltzmann law, Kepler's laws, Doppler shift formula, blackbody
radiation, and Newtonian gravity.
```

DeepSeek-V3.2 formula slot. In a representative DeepSeek-V3.2 run, the formula slot learned a stricter numerical-output policy. This slot is especially aligned with the Formula benchmark, where the input gives an explicit formula and the evaluator expects a compact numeric answer.

```
You are a finance calculation assistant. Your task is to compute the
answer using the formula and data provided in the user's question,
then output only the numeric result with exactly two decimal places
and no other text.

Procedure:
1. Identify the formula given in the user's message.
2. Identify the explanation of each variable in the formula.
3. Extract the numeric values for each variable from the question text.
   Convert all percentage inputs into their decimal equivalent.
4. Substitute the values into the formula and perform the calculation.
5. If the formula calculates a financial rate, yield, return, or cost,
   do not multiply the decimal result by 100.
6. Round the final result to two decimal places.
7. Output only the resulting number, with no words, units, labels,
   currency symbols, or percentage signs.
```

A.2 Representative Raw Outputs

Formula: unit scale and sign. For a free-cash-flow computation, the required operation is operating cash flow minus capital expenditure. The target is a dollar-scale scalar value. The empty response flips the sign and keeps the million-scale decimal:

```
-0.40
```

The learned response applies the provided formula at the correct scale and emits the strict numeric answer:

```
Answer: 400000.00
```

This is representative of the Formula gains: the learned prompt pairs formula application with unit-scale discipline and a parseable final representation.

HumanEval: executable continuation. For a HumanEval function-completion task, the required behavior is to sum even elements at odd indices and return the sum. The empty response captures the expression but does not produce an executable function body:

```
sum(lst[i] for i in range(1, len(lst), 2) if lst[i] % 2 == 0)
```

The learned response preserves the logic while satisfying the continuation contract:

```
    total = 0
    for i in range(1, len(lst), 2):
        if lst[i] % 2 == 0:
            total += lst[i]
    return total
```

This is representative of the HumanEval gains: the learned behavior combines implementation logic with the executable contract expected by the evaluator.

GPQA Diamond: domain knowledge underweighted after stronger reasoning. For an exoplanet-density question, the target is the higher-mass Earth-composition planet. The empty response uses a domain-specific mass-radius prior:

```
For rocky planets of similar composition, radius grows sublinearly with
mass because stronger gravity compresses the material. A five-Earth-mass
rocky planet is therefore denser than Earth, while a half-Earth-mass
rocky planet is less dense.

Answer: the higher-mass Earth-composition option.
```

The learned response performs a more explicit density-formula comparison, but it treats equal composition as if it implied equal density:

```
Density is mass divided by volume. For a spherical planet,
rho = M / (4/3 pi R^3). If composition is the same, density is
constant; the radius scales with mass so that all Earth-composition
options remain approximately Earth density. Since the Earth-mass,
Earth-radius option is exactly Earth-like, choose that option.
```

The error is not a lack of reasoning steps. The learned prompt induces a stronger generic reasoning pattern, but the reasoning is grounded in the wrong physical prior: same composition does not imply constant density for rocky exoplanets, because self-compression changes the mass-radius relation.

B Additional Analyses

This appendix reports studies that extend the main experiments. Unless noted otherwise, the target model is Qwen3-4B-Instruct and the protocol matches Appendix C.

B.1 Equal-Budget Routing Controls

Because the four benchmarks have clear domain boundaries, a simple dataset-identity router might already recover most of the benefit that Table 2 attributes to EEVEE. We therefore add four external controls under a matched budget, so that every routed variant evolves the same total amount of prompt text and the rows differ only in how inputs are assigned to slots: a mixed single prompt, random routing, a supervised dataset-ID classifier trained on adaptation-train labels, and an oracle router handed the true source dataset.

Table 6: Equal-budget routing controls on Qwen3-4B-Instruct, sorted by average score.

Routing design	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
No adaptation	56.00	45.22	14.79	49.46	41.37
Random router	28.62	44.17	25.39	43.90	35.52
Mixed single prompt	35.69	55.33	28.67	48.78	42.12
Oracle dataset ID	53.20	42.33	23.58	69.51	47.15
Learned classifier	53.87	40.67	**29.88**	67.48	47.98
EEVEE	**54.55**	**54.55**	25.27	**72.63**	**51.75**

Dataset identity is easy to recover: the classifier reaches **100% dataset-ID accuracy**. Yet it averages 47.98 and the oracle 47.15, both below EEVEE's 51.75, and EEVEE exceeds the oracle on all four benchmarks. Random routing collapses to 35.52, below no adaptation at all, so added prompt capacity explains nothing on its own. Neither straightforward dataset separation nor extra prompt text accounts for the gains; the dataset-ID oracle is a source-identification control, not an upper bound on prompt optimization.

Table 7 shows why beating the oracle is even possible: the learned router does not reproduce the dataset partition. TheoremQA is split across all four slots, and most of it shares a slot with Formula rather than forming its own. The router groups inputs by the capability and output behavior they require, pooling examples across benchmark boundaries so that each prompt receives more coherent feedback than any dataset-aligned assignment gives it. That partition is not fixed in advance either: as a slot's prompt evolves, so does the behavior the slot encodes, which is what separates co-evolution from routing over a fixed set of prompt experts.

Table 7: The partition the router actually learns: percentage of each benchmark's test items sent to each slot in one representative run. Slot labels summarize the behavior its learned prompt encodes; dashes are zero.

Learned slot behavior	GPQA Diamond	Formula	TheoremQA	HumanEval
Formula use, numeric output format	–	**100.0**	**62.4**	–
Scientific multiple choice	**100.0**	–	11.6	–
Math/combinatorics, "Solution:" contract	–	–	23.9	–
Function completion and execution	–	–	2.1	**100.0**

B.2 Online Test-then-Adapt Evaluation

This experiment removes the pre-built adaptation split entirely. All examples form one fixed stream; each is scored the first time it is encountered and can influence only subsequent updates, so no external adaptation/test split exists and no future example is ever visible. *Cumulative* updates from every example observed so far, whereas *new-window* updates from

only the most recent window; both retain the previous router and prompts rather than restarting from scratch.

Table 8: Strict prequential test-then-adapt evaluation on Qwen3-4B-Instruct.

Method	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
ACE online	29.12	42.23	16.92	28.86	29.28
EEVEE cumulative	**51.01**	50.67	**24.71**	47.97	43.59
EEVEE new-window	50.00	**52.95**	23.44	**49.49**	**43.97**

Both variants beat ACE on all four benchmarks, improving the average by 14.31 and 14.69 points.

B.3 Adaptation-Phase Cost Audit

Figure 5 reports the per-example cost paid at test time. Table 9 reports the one-off adaptation cost that precedes it, covering router evolution, prompt evolution, reflection, candidate evaluation, and validation.

Table 9: Adaptation-phase cost on Qwen3-4B-Instruct, with per-example test cost repeated for reference.

Method	Adaptation calls	Adaptation tokens	Wall-clock	Test tokens/example
ACE	3,529	58.36M	2.90 h	21.30k
GEPA	77,715	226.81M	8.59 h	**3.47k**
EEVEE	104,460	154.71M	7.86 h	4.32k

Compared with GEPA, EEVEE uses 31.8% fewer adaptation tokens at similar wall-clock time, though it makes more and shorter calls. ACE adapts far more cheaply but deploys a 4.9× larger test-time context, so its cost is paid again on every query rather than once.

B.4 Slot-Count Sensitivity

We vary only the number of prompt slots K, holding the model, data, budget, and evaluation fixed. Routing is vacuous at $K = 1$, so that setting is the mixed single-prompt control from Table 6, which spends the entire budget on prompt evolution.

Table 10: Four-benchmark average as a function of the number of prompt slots.

Slots	$K = 1$	$K = 2$	$K = 4$	$K = 6$
Avg.	42.12	42.04	**51.75**	48.74

Too few slots substantially reduce performance: $K = 1$ and $K = 2$ land near the unadapted baseline, confirming that specialization, not extra prompt text, is what the method depends on. The close $K = 4$ and $K = 6$ results indicate robustness to moderate changes in slot count.

B.5 Robustness to Sparse and Noisy Feedback

EEVEE assumes an observed correctness signal, so we test what happens when that signal is incomplete or wrong. *Sparse-p* provides correctness feedback for only p% of adaptation examples; the remainder yield no evolution signal. *Noisy-p* flips the binary correctness signal for p% of adaptation examples, leaving everything else unchanged. Each resulting system is evaluated on the same held-out questions from the four benchmarks, averaged over three repeats, and reported as the fraction of clean-feedback performance retained.

EEVEE retains 88.4–89.7% of clean performance with feedback on only a quarter to a half of adaptation examples, and 85.7–89.9% when a tenth to a fifth of the signals are corrupted. Sparse-25 scoring marginally above Sparse-50 is within run-to-run variation (Appendix C.2);

Table 11: Clean-feedback performance retained under degraded feedback.

Feedback condition	Sparse-50	Sparse-25	Noisy-10	Noisy-20
Retained performance	88.4%	89.7%	89.9%	85.7%

the larger drop at Noisy-20 follows the expected noise-strength trend. Delayed feedback can be folded in when it arrives, and in the absence of any reliable signal EEVEE keeps its current router and prompts rather than updating without evidence.

B.6 Mixture Composition and Task Order

To test sensitivity to how the adaptation stream is composed, we hold the test set fixed and vary only the task composition of a 200-example adaptation set, split 100/100 into internal train and validation. This adaptation set is smaller than the one used for the main runs, so absolute scores sit below the 51.75 of Table 1; only the comparison across mixtures is meaningful.

Table 12: Four-benchmark average under different adaptation-set mixtures, given as GPQA/TheoremQA/Formula/HumanEval example counts.

Mixture	50/50/50/50	80/40/40/40	40/80/40/40	40/40/80/40	40/40/40/80
Avg.	48.59	40.79	48.18	**50.71**	50.53

Performance is stable across most mixtures. The GPQA-heavy mixture is the clear outlier, consistent with the knowledge-heavy limitation discussed in Section 3.7: over-weighting the one task where learned prompts help least also degrades the rest. For task order, shuffled and blocked streams over identical examples obtain 49.05 and 50.19. Because EEVEE canonicalizes eligible example IDs before seeded minibatch sampling, stored order does not systematically change which data drive optimization, and the similar scores confirm this.

B.7 Cross-Model Router Ablations

Table 2 covers Qwen3-4B-Instruct only. We repeat the router ablations on DeepSeek-V3.2 under matched budgets; the DeepSeek router variants use three evaluations per test example, while the baseline and full EEVEE rows reproduce the averages from Table 1.

Table 13: Router ablations on both target models.

Model	Variant	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
Qwen3-4B-Instruct	Baseline	56.00	45.22	14.79	49.46	41.37
	Default Router	47.03	47.11	26.10	54.06	43.58
	Manual Router	35.24	42.11	23.96	47.42	37.18
	EEVEE	54.55	54.55	25.27	72.63	**51.75**
DeepSeek-V3.2	Baseline	64.98	30.00	21.21	42.82	39.75
	Default Router	44.78	57.83	32.85	90.24	56.43
	Manual Router	58.59	56.17	39.94	89.02	60.93
	EEVEE	63.08	60.55	39.84	92.82	**64.07**

The stronger model substantially improves fixed routing: the manual router rises from 37.18 on Qwen3-4B-Instruct to 60.93 on DeepSeek-V3.2. Full EEVEE nevertheless remains best at 64.07, so better intrinsic routing ability narrows but does not remove the benefit of router–prompt co-evolution.

B.8 Inference-Time Gating for Knowledge-Heavy Tasks

Because learned prompts can hurt closed-book QA, we test two safeguards that require no re-running of evolution. *Selection 2-of-3* applies an evolved prompt only when it receives at least two of three router votes, and otherwise falls back to the empty prompt. The *harm-aware* gate is a TF-IDF logistic regression that predicts whether the selected prompt is likely to hurt the input, using only the question and prompt text and no dataset identity. We evaluate on MMLU-Pro to measure knowledge protection and on MBPP to check that beneficial prompts are not rejected wholesale. Both gates use the completed routers and prompts from three independent EEVEE runs.

Table 14: Inference-time gating on held-out tasks. MMLU-Pro rows are a matched first pass, so they differ slightly from the three-repeat averages in Table 3.

Held-out task	Empty	EEVEE	Harm-aware	Selection 2-of-3
MMLU-Pro, matched first pass	70.65	68.81	69.48	**70.24**
MBPP, three-repeat average	69.29	**70.42**	70.33	70.09

On MMLU-Pro the harm-aware gate and Selection recover 0.67 and 1.43 points, leaving Selection only 0.41 below the empty prompt. On MBPP, where evolved prompts help, the gates retain 70.33 and 70.09 against the ungated 70.42. Gating therefore recovers most of the knowledge-task degradation without collapsing into blanket rejection of evolved prompts.

B.9 Adaptation Cost Scaling

We measure how the adaptation budget in Figure 5 grows along three axes. Expanding the adaptation pool from 200 to 681 examples (3.41×) raises model calls from 34,072 to 107,233 (3.15×) and tokens from 48.33M to 155.31M (3.21×), so cost is approximately linear in example count. Across 200-example mixtures, totals range from 37.90M to 59.46M tokens, so sequence length and task composition move the constant factor but not the trend. Increasing slots from one to six raises tokens by only 10.7% (140.78M to 155.84M), because global budgets are shared across slots and validation examples are partitioned among them rather than duplicated. With examples and slots fixed, task diversity introduces no separate per-task loop, so if each additional task contributes a fixed number of examples the dominant cost again grows approximately linearly with task count.

Compute. All experiments call hosted or served API endpoints and train no local weights. The additional studies reported in this appendix consumed approximately 1.1K H200 GPU-hours in total.

C Reproducibility and Experimental Details

This section records the core settings used for the main four-benchmark experiments in Table 1. The benchmark suite consists of GPQA Diamond Rein et al. (2023), Formula Wang et al. (2025), TheoremQA Chen et al. (2023), and HumanEval Chen et al. (2021). The held-out generalization study uses MBPP Austin et al. (2021) and MMLU-Pro Wang et al. (2024) after prompt learning on the four primary benchmarks. The single-benchmark diagnostic also includes FiNER Loukas et al. (2022) and IFBench Pyatkin et al. (2025). For each benchmark, we cap the benchmark size at 500 examples, split examples into train and test partitions with a 0.5/0.5 split, reserve half of the training partition as validation data, exclude train and validation examples from the final test set, and use data seed 42.

The main target models are Qwen3-4B-Instruct Yang et al. (2025) and DeepSeek-V3.2 DeepSeek-AI et al. (2025) in non-thinking mode. For the Qwen3-4B-Instruct runs, all model roles use the same Qwen endpoint with temperature 0.7, top-p 0.8, and a maximum generation length of 16,384 tokens. For the DeepSeek-V3.2 non-thinking runs, the endpoint uses temperature 1.0, top-p 0.95, maximum generation length 8,192 tokens, and disabled thinking. The main benchmark configurations use rule-based benchmark judges when available.

Evolution settings. Each main run retains four bootstrap prompts. Bootstrap prompt evolution uses a budget of 10 candidate steps. The router–prompt co-evolution phase uses a total mini-step budget of 150, router and prompt windows of 3, and phase-switch threshold 0.005. The router score during evolution combines downstream score, routing consistency, and balance with weights 0.6/0.2/0.2; the final router selection uses downstream score only, with weights 1.0/0.0/0.0. Prompt evolution uses a final per-label budget of 60, minibatch size 5, and 4 parallel prompt slots.

Execution settings and resources. The Qwen3-4B-Instruct main runs use 50 API workers for evaluation, router calls, and final testing. The DeepSeek-V3.2 non-thinking main runs use 10 API workers for the same roles. All reported final tests use three repeats, a maximum prompt length of 2,000 tokens, at most 3 retries, and at most 5 logged examples for diagnostic traces. The experiments call hosted or served API endpoints and do not train or fine-tune local model weights; accordingly, no local GPU training resource is required. Wall-clock time is backend- and rate-limit-dependent, so the paper reports token usage instead. In Figure 5, EEVEE uses 4.32k total tokens per test example on average, compared with 3.47k for GEPA and 21.30k for ACE.

Reproducibility scope. The paper is empirical and does not present formal theoretical results or proofs. Reproducing the main claims requires reproducing the mixed-benchmark adaptation protocol, model endpoints or comparable served checkpoints, benchmark splits, router–prompt evolution settings, and final-test evaluation described above. Because router and prompt evolution are stochastic, exact routers and prompt texts may vary across runs even under the same settings; Table 16 reports this run-to-run variation for the main average score.

C.1 Hyperparameter Robustness

We test whether EEVEE is sensitive to the specific router-score and prompt-search hyperparameters used in the main experiments. For each hyperparameter configuration, we run three independent trials, first average the three trials within that configuration, and then compare the resulting eight configuration-level means. The configurations vary the annealing target for router scores, the consistency and balance weights, and the final prompt-search budget, minibatch size, and temporary prompt-pool size.

Table 15: Hyperparameter robustness on Qwen3-4B-Instruct. Scores are percentages; each row is the mean over three independent runs for one configuration. Avg. is the macro average over the four benchmarks.

Configuration	GPQA Diamond	Formula	TheoremQA	HumanEval	Avg.
No-decay annealing	49.94	52.17	25.66	58.67	46.61
Soft-landing annealing	45.01	57.44	21.33	69.24	48.26
Strong-regularizer annealing	45.34	46.61	24.57	63.69	45.05
Classic reference	44.78	60.06	26.20	57.99	47.26
Prompt search tighter	47.92	53.33	27.41	63.01	47.92
Prompt search wider	48.15	62.39	23.94	61.38	48.96
Consistency-heavy weights	51.74	50.33	26.79	63.96	48.20
Balance-heavy weights	49.16	54.89	26.93	72.90	50.97

The eight configuration-level averages range from 45.05 to 50.97, a span of 5.92 points, with a sample standard deviation of 1.73 points across configurations. Thus the aggregate result is stable under the tested hyperparameter perturbations. Individual benchmarks can move more than the average, but no configuration collapse is observed, and every configuration improves over its corresponding initial-empty baseline in macro average.

C.2 Main-Result Variation Across Runs

Table 16 reports the mean and sample standard deviation of the main average score over the three independent runs used in Table 1. The average score is stable for EEVEE: the standard deviation is 1.62 points on Qwen3-4B-Instruct and 1.08 points on DeepSeek-V3.2. Individual benchmark scores can still differ more noticeably across runs. The reason is that router evolution is stochastic and can discover different routing policies; different policies allocate examples to different prompt slots, which changes which prompt behaviors receive the most feedback and can shift per-task scores even when the overall average remains stable.

Table 16: Mean and sample standard deviation of the main average score over three runs.

Target model	Method	Average score
Qwen3-4B-Instruct	Baseline	41.37 ± 0.51
	ACE	34.92 ± 4.14
	GEPA	37.73 ± 1.73
	EEVEE	$\mathbf{51.75 \pm 1.62}$
DeepSeek-V3.2	Baseline	39.75 ± 1.52
	ACE	49.83 ± 2.07
	GEPA	55.83 ± 4.48
	EEVEE	$\mathbf{64.07 \pm 1.08}$

D Ethics, Assets, and LLM Usage

Human subjects and privacy. The experiments use public benchmarks and model API calls. We do not collect new human-subject data, run crowdsourcing studies, or introduce a dataset containing personal information.

Existing assets and code availability. The experiments use public benchmarks, public or provider-served model checkpoints, and published prompt-learning baselines such as GEPA and ACE. Code, configuration files, reproduction scripts, and asset metadata are available at https://github.com/Princeton-AI2-Lab/EEVEE, with an overview at https://princeton-ai2-lab.github.io/EEVEE/. This work introduces a method rather than a new dataset or model checkpoint.

Responsible use and broader impacts. EEVEE can make heterogeneous-task prompt adaptation more efficient by reducing the need to maintain one prompt-learning run per task family. The main practical risk is that noisy, incomplete, or distribution-shifted feedback can reinforce incorrect heuristics. Adapted prompts should be validated on held-out data before deployment, and benchmark gains should not be interpreted as deployment reliability guarantees.

LLM usage. LLMs are core components of the method: they act as the target model being adapted and as the prompt researcher, prompt reflector, router selector, router researcher, router reflector, router reasoner, and evaluator where applicable. In the ablation study, GPT-5.4 was used once to write a fixed manual router, which was then held constant for evaluation. LLMs were also used for language editing and formatting of the manuscript; this editing use did not change the scientific claims, experimental data, or conclusions.