FROM A PROMPT TO REPERTOIRES: EVOLVING FUNCTIONAL REPERTOIRES ENABLE LLM CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning remains challenging for large language models, which must enable models to acquire new skills and knowledge without degrading existing capabilities. Existing approaches typically address this challenge by carefully designing how model parameters are updated. In contrast, prompt optimization avoids costly parameter updates while achieving competitive or even superior performance to reinforcement learning methods such as GRPO on individual knowledge-intensive and reasoning tasks. This raises a natural question: *Can prompt optimization, as an efficient adaptation approach, be directly applied to continual learning?* Our analysis shows that, under sequential task adaptation, it suffers from catastrophic forgetting, while optimized prompts accumulate rules that overfit to local task distributions. To address these limitations, we propose *Evolving Functional REpertoires* (EFRE), which replaces a single prompt with a repertoire of functions that evolves as new tasks arrive: compatible updates refine existing functions, while conflicting updates trigger the emergence of new ones. On a three-task continual-learning stream, EFRE achieves a final average performance 7.50 percentage points higher than GRPO. Moreover, after adaptation to the Bio task, its performance on FinQA decreases by only 1.56 percentage points, compared with 25.10 percentage points for the base prompt optimization method. We further instantiate EFRE in a minimal agent system and observe consistent improvements across different backbone models. Overall, these results demonstrate EFRE's strong performance in continual learning for large language models and highlight its substantial potential for continual learning in advanced agent systems.

1 INTRODUCTION

Continual learning is becoming increasingly important for large language models (LLMs), which need to continually acquire new knowledge and skills while preserving what they have already learned (Harrington et al., 2026; Asawa et al., 2026). Most existing approaches address this challenge by controlling how model parameters are updated, for example through low-rank adaptation (LoRA) (Hu et al., 2021), replay-based training (Lopez-Paz & Ranzato, 2017), or reinforcement-learning-based optimization such as GRPO (Shao et al., 2024). Although effective, these methods incur substantial training overhead. Enabling efficient continual learning remains an open challenge.

Recently, prompt optimization (Pryzant et al., 2023; Yuksekgonul et al., 2025; Agrawal et al., 2026; Zhang et al., 2026) has emerged as an efficient approach to adapting LLMs to downstream tasks without parameter updates. These methods have achieved strong performance on knowledge-intensive and reasoning tasks, in some cases outperforming reinforcement learning methods such as GRPO. This naturally raises the question: *Can prompt optimization, as a powerful and efficient optimization method, be directly applied to continual learning?* Our analysis and prior studies (Harrington et al., 2026) suggest otherwise: under sequential multi-task learning, repeated prompt updates overwrite previously learned knowledge and accumulate case-specific rules rather than generalizable principles, limiting their effectiveness in continual learning.

Motivated by these limitations, we introduce *Evolving Functional REpertoires* (**EFRE**), a continual learning method that evolves both the capabilities and composition of a functional repertoire as new

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

learning tasks arrive. Each function is represented by an optimized prompt. Historical consistency checks guide this evolution: compatible updates refine existing functions and enable reuse across tasks, while updates that cannot be absorbed without compromising historical capabilities trigger the emergence of new functions. Through this process, functional specialization and reuse arise automatically, reducing interference between new and previously learned capabilities.

We evaluate EFRE on the ToolUse (Tang et al., 2023) $\rightarrow$ FinQA (Chen et al., 2021) $\rightarrow$ SciKE-Bio (Feng et al., 2024) continual-learning sequence and several single-task benchmarks. With Qwen3-8B (Qwen Team, 2025a), EFRE achieves a final average score of 58.75, outperforming GRPO (Shao et al., 2024) and GEPA (Agrawal et al., 2026), while limiting forgetting on previously learned tasks after Bio to 1.57 points, versus 8.32 for GRPO and 11.82 for GEPA. EFRE also consistently improves over prompt-optimization and GRPO baselines on single-task benchmarks. In a minimal agent system spanning six continual-learning tasks, EFRE similarly outperforms ICL (Brown et al., 2020) and GEPA: it attains average scores of 18.0 with Qwen3-8B (vs. -3.5 and 9.5) and 36.3 with GPT-5.6-Luna (OpenAI, 2026) (vs. 22.2 and 29.5). Together, these results demonstrate EFRE's effectiveness for both standard and agent continual learning.

Our key contributions are threefold:

- We identify two key limitations of prompt optimization in continual learning: updates overwrite previously learned knowledge, causing forgetting, and prompts accumulate case-specific rules rather than generalizable principles.
- We introduce EFRE, a non-parametric continual learning method whose functional repertoire evolves with incoming tasks through the refinement and reuse of existing functions and the emergence of new ones, guided by historical consistency checks.
- We empirically demonstrate that EFRE improves both continual learning and single-task optimization, outperforming existing prompt optimization methods and GRPO in overall performance. Further gains in agent continual-learning settings highlight its potential for continual learning in real-world intelligent systems.

2 Related Work

Continual Learning. Continual learning is a learning paradigm that requires a model to acquire new capabilities while preserving previously learned knowledge and behaviors. Existing approaches mainly address the stability–plasticity trade-off through replay (Lopez-Paz & Ranzato, 2017), regularization (Kirkpatrick et al., 2017), parameter isolation (Rusu et al., 2016), parameter-efficient tuning (Hu et al., 2021), and reinforcement-learning-based optimization (Shao et al., 2024). Recent work has further extended continual learning to LLMs and language agents, where models must continually adapt to new tasks, knowledge, and interaction environments (Shenfeld et al., 2026; Asawa et al., 2026). EFRE studies this problem from a non-parametric perspective, maintaining and evolving external functional contexts rather than continually updating model parameters.

Non-Parametric Optimization. Non-parametric optimization is a class of lightweight optimization methods that improves model behavior without updating model parameters, with prompt optimization being a representative approach (Pryzant et al., 2023; Yuksekgonul et al., 2025). Existing methods optimize prompts, contexts, memories, or other external textual states using task feedback, reflection, or iterative search (Shinn et al., 2023; Agrawal et al., 2026; Zhang et al., 2026). Recent prompt optimization methods have demonstrated strong performance across reasoning, knowledge-intensive, and agent tasks, showing that textual states can serve as effective optimization targets (Agrawal et al., 2026; Zhang et al., 2026). Building on this, EFRE organizes the evolving non-parametric learning state into a repertoire of functional contexts, enabling prompt optimization to support continual learning across sequential tasks.

3 Methodology

In this section, we introduce *Evolving Functional REpertoires* (EFRE) . We first formulate continual learning with frozen LLMs. We then describe prompt optimization for continual learning and discuss its limitations. Finally, we present EFRE, which uses historical consistency checks to guide the refinement of existing functions and the emergence of new ones as learning tasks arrive (Figure 1).

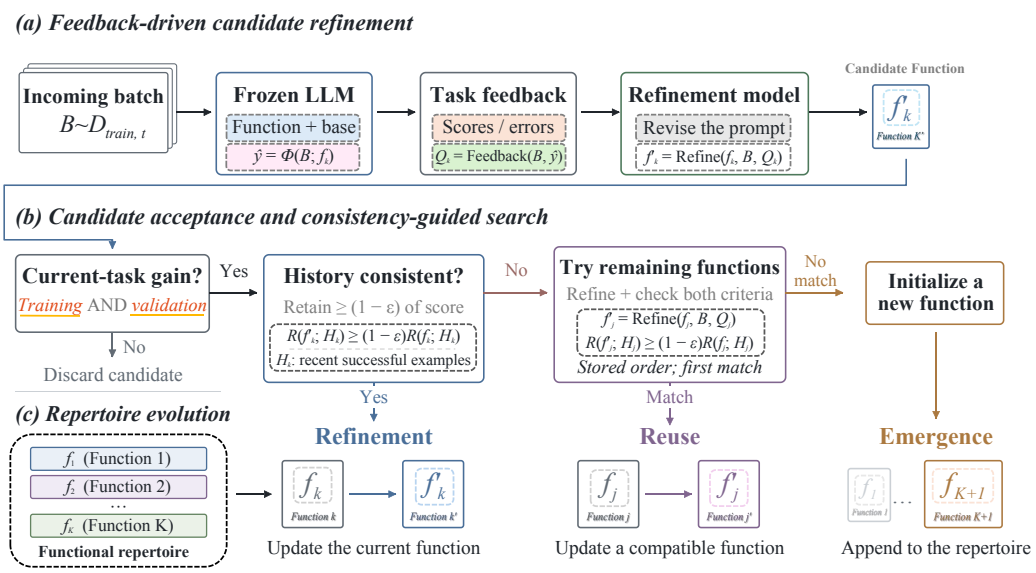

Figure 1: Overview of Evolving Functional REpertoires (EFRE). Each function is represented by a prompt and evolves through feedback-driven refinement on incoming data. Candidates that improve current performance undergo a historical consistency check: compatible updates refine the current function, while incompatible updates trigger a sequential search for another function that can absorb them. If no compatible update is found, a new function emerges. The repertoire thus evolves through functional refinement, reuse, and emergence, enabling continual adaptation while preserving previously learned capabilities.

3.1 Problem Statement

We study continual learning for a pretrained language model Φ whose parameters remain fixed. At stage t, the learner receives a training batch $D_{\text{train},t}$ and updates its external textual state through $S_t = \mathcal{U}(S_{t-1}, D_{\text{train},t})$, where $\mathcal{U}$ is the update procedure. The state may contain prompts, skills, or other textual information that guides the model's behavior. The objective is to acquire new capabilities while preserving those learned at earlier stages.

For an evaluation set D_{eval}, we define performance as:

$$R(S; D_{\text{eval}}) = \text{Eval}(\Phi(\cdot; S), D_{\text{eval}}), \tag{1}$$

where Eval scores the model conditioned on S, with higher values indicating better performance. We use the notation R when scoring training, validation, and history data. For continual-learning evaluation, $D_{\text{eval},i}$ denotes the held-out evaluation set for stage i and is never used to update S.

Plasticity and Stability. We characterize each update through two classical properties of continual learning. *Plasticity* measures its performance gain on the current task:

$$\mathcal{J}^{(t)}_{\text{plas}} = R(S_t; D_{\text{eval},t}) - R(S_{t-1}; D_{\text{eval},t}). \tag{2}$$

A larger value indicates more effective learning from the incoming data. *Stability* measures the average performance change on previously encountered tasks:

$$\mathcal{J}^{(t)}_{\text{stab}} = \frac{1}{t-1} \sum_{i=1}^{t-1} \left[R(S_t; D_{\text{eval},i}) - R(S_{t-1}; D_{\text{eval},i}) \right], \; t > 1. \tag{3}$$

Negative values indicate average forgetting, zero indicates no net change in historical performance, and positive values indicate beneficial backward transfer (Lopez-Paz & Ranzato, 2017). Effective continual learning should achieve positive plasticity while limiting degradation on previous tasks.

3.2 PROMPT OPTIMIZATION FOR CONTINUAL LEARNING

In a continual learning setting, prompt optimization implements $S_t = \mathcal{U}(S_{t-1}, D_{\text{train},t})$ by rewriting the prompt represented by S_{t-1}. At stage t, the frozen model runs on $D_{\text{train},t}$ using the current prompt. The task environment returns feedback $\mathcal{Q}_t$, including performance scores and, when available, textual feedback such as test results, error messages, or explanations of incorrect answers. An external refinement model M_{refine} uses this feedback and the training data to revise S_{t-1}:

$$S'_t = M_{\text{refine}}(S_{t-1}, D_{\text{train},t}, \mathcal{Q}_t). \tag{4}$$

The candidate replaces the current prompt only if it improves performance on both the training batch and a separate validation set:

$$\begin{aligned} R(S'_t; D_{\text{train},t}) &> R(S_{t-1}; D_{\text{train},t}), \\ R(S'_t; D_{\text{val},t}) &> R(S_{t-1}; D_{\text{val},t}). \end{aligned} \tag{5}$$

Otherwise, $S_t = S_{t-1}$. The validation set $D_{\text{val},t}$ is used for update selection and is distinct from the held-out evaluation set $D_{\text{eval},t}$.

Our analysis identifies limitations in both stability and plasticity. The acceptance rule checks current performance but does not explicitly assess retention on previous tasks. Repeated rewriting can therefore overwrite instructions that support previously learned capabilities, causing catastrophic forgetting. Meanwhile, prompts can accumulate case-specific rules rather than generalizable principles, making new requirements difficult to accommodate. These limitations motivate EFRE, which checks historical consistency before accepting updates and allows functions to emerge when updates cannot be incorporated into existing functions without compromising historical performance.

3.3 EFRE: EVOLVING FUNCTIONAL REPERTOIRES

We first define functions and their associated repertoire, then describe how the repertoire evolves during continual learning and supports inference.

Functions and Functional Repertoire. A *function* f_k is a prompt encoding a learned capability, such as a reasoning procedure or instructions for handling particular inputs. A *functional repertoire* $\mathcal{F}_t = \{f_1, \ldots, f_{K_t}\}$ contains the K_t functions available at stage t. Each function maintains a history $D_{\text{history},k}$ of at most H recent examples successfully absorbed by that function. Functions are not assigned to tasks in advance: one function may serve multiple tasks, and one task may involve multiple functions. As learning proceeds, both individual functions and the composition of the repertoire evolve. For local evaluation, $R(f_k; D)$ denotes the performance of the frozen LLM Φ on dataset D, directly conditioned on f_k and shared base instructions p_{base}.

Repertoire Evolution and Inference. EFRE starts with one function and subsequently attempts each update on the branch accepted by the most recent successful update. At stage t, the frozen LLM runs on $D_{\text{train},t}$ using the current function f_k. The environment returns feedback $\mathcal{Q}_{t,k}$, such as correctness scores, test results, or execution errors, which the refinement model uses to propose

$$f'_k = M_{\text{refine}}(f_k, D_{\text{train},t}, \mathcal{Q}_{t,k}). \tag{6}$$

Following Equation 5, the candidate must improve performance on both $D_{\text{train},t}$ and $D_{\text{val},t}$. If either condition fails, it is discarded without evaluating other functions.

A candidate that improves current performance must also pass a *historical consistency check*:

$$R(f'_k; D_{\text{history},k}) \geq (1 - \epsilon)\, R(f_k; D_{\text{history},k}), \tag{7}$$

where scores are nonnegative and $\epsilon \in [0, 1]$ specifies the maximum proportional decrease in historical performance. For example, $\epsilon = 0.1$ requires retention of at least 90% of the original score. The check is skipped when the history is empty. A passing candidate replaces the current function, and successfully absorbed examples are appended to its history, retaining the most recent H examples.

If the candidate improves current performance but fails historical consistency, EFRE searches the remaining functions in their stored order. For each function, it obtains feedback on the incoming batch and generates a revision using Equation 6. The search stops at the first candidate that improves both training and validation performance relative to its original function and passes Equation 7 on

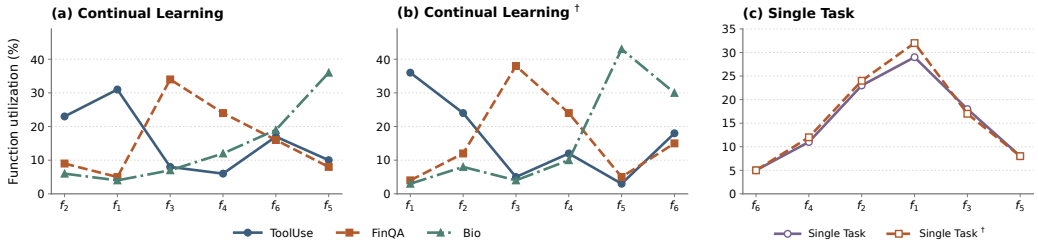

Figure 2: Functional reuse patterns in continual-learning and single-task settings. Curves show utilization of the six most-used functions. (a), (b) report utilization on ToolUse, FinQA, and SciKE-Bio. (c) compares both refinement configurations for single-task optimization, with a separate vertical scale. † denotes GPT-5.6-Luna as the refinement model; Qwen3-8B is used otherwise.

that function's own history. Only the matched function and its history change; the original branch remains unchanged. This enables *functional reuse* through compatible updates from different tasks.

If no compatible update is found, the refinement model proposes a new function using the incoming data and feedback from the failed attempts. The new function is added only if it improves training and validation performance relative to the current prompt before this update. Its history is initialized with up to H examples that it successfully handles. We call this automatic creation of a new function *functional emergence*. The repertoire can contain at most 50 functions; when this limit is reached, an update that cannot be absorbed by an existing function is discarded. After an accepted refinement, reuse, or emergence, the accepted branch becomes the current function for the next update. The repertoire thus evolves by incorporating compatible capabilities into existing functions and creating new ones when historical incompatibility prevents reuse.

At inference time, a textual selector σ receives the input and current function prompts, selects a function, and directs the frozen LLM to generate a response:

$$k(x) = \sigma(x; \mathcal{F}_t), \;\; \hat{y} = \Phi\big(x; p_{\text{base}}, f_{k(x)}\big)\,. \tag{8}$$

The selector uses fixed instructions; its available function prompts change as the repertoire evolves. No task labels are required. The shared instructions, repertoire, histories, and selector constitute the external state S_t. Learning updates this textual state while keeping the parameters of Φ fixed. Further methodological details are provided in Appendix D.

4 Experiments

We evaluate EFRE on the ToolUse $\rightarrow$ FinQA $\rightarrow$ SciKE-Bio sequence to assess its continual-learning performance. We conduct ablation studies and mechanism analyses to examine historical consistency, function selection, and functional specialization and reuse. Finally, we evaluate EFRE on benchmarks such as HotpotQA to assess its effectiveness in single-task optimization.

4.1 Experimental Setup

Benchmark Datasets. We construct a continual-learning sequence using ToolUse (Tang et al., 2023), FinQA (Chen et al., 2021), and SciKE-Bio (Feng et al., 2024). After each learning stage, we evaluate performance on the current and previously encountered tasks. We additionally conduct single-task experiments on HotpotQA (Yang et al., 2018), IFBench (Pyatkin et al., 2025), AIME-2025 (Mathematical Association of America, n.d.), and LiveBench-Math (White et al., 2025), and evaluate continual learning in agent settings using Continual Learning Bench (Asawa et al., 2026). We report the benchmark-specific evaluation metrics. For continual-learning experiments, we also measure plasticity and stability as defined in Section 3.1, which quantify improvement on the current task and changes in performance on previous tasks, respectively.

Comparing Baselines. We compare EFRE with methods that adapt model parameters or external textual states. Among parameter-based methods, SFT learns directly from demonstrations through supervised fine-tuning. SDFT (Shenfeld et al., 2026) performs on-policy self-distillation using the

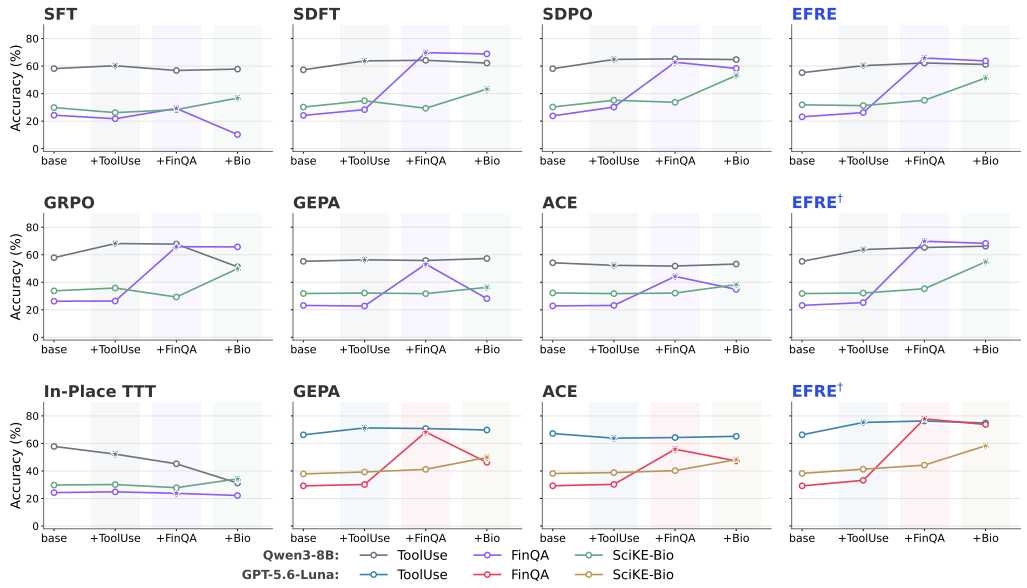

Figure 3: Continual-learning performance along the ToolUse → FinQA → SciKE-Bio sequence. Each panel shows one method, with curves tracking performance on all three tasks before adaptation and after each learning stage. Stars mark performance immediately after training on the corresponding task, and shaded regions indicate the task introduced at each stage. The two color palettes distinguish Qwen3-8B and GPT-5.6-Luna backbones. EFRE variants are grouped in the rightmost column; EFRE uses Qwen3-8B for refinement by defualt; † indicates GPT-5.6-Luna instead.

Table 1: Functional organization of the learned repertoires. For continual learning, **Cross-task reuse** denotes the percentage of evaluation instances routed to functions used by at least two tasks, averaged equally across ToolUse, FinQA, and SciKE-Bio. **Shared Func.** counts functions used by at least two tasks, while **Func.** denotes total repertoire size. Cross-task reuse is not defined for single-task settings, where each benchmark is optimized independently. The final column illustrates a learned function. Rows marked with † use GPT-5.6-Luna for refinement; others use Qwen3-8B.

Setting	Cross-task reuse	Shared Func.	Func.	Example learned function
Continual Learning	41.7%	2	7	Identify the next required action, select a tool, and check its arguments against the tool schema before execution.
Continual Learning†	**56.3%**	**6**	19	Extract the relevant financial values and units, write the calculation, and verify its result against the source table.
Single-task	–	–	8	Decompose the question into two linked subquestions, gather evidence, and synthesize a supported answer.
Single-task†	–	–	12	Translate the conditions into equations, solve them, and verify candidates against the original constraints.

model conditioned on demonstrations as its teacher. SDPO (Hübotter et al., 2026) distills predictions from a feedback-conditioned self-teacher into the policy. GRPO (Shao et al., 2024) performs reinforcement learning using relative rewards within groups of sampled responses. In-Place TTT (Feng et al., 2026) adapts selected MLP projection weights during inference. Among methods that keep model parameters fixed, GEPA (Agrawal et al., 2026) optimizes prompts through natural-language reflection on execution traces and evaluation feedback, while ACE (Zhang et al., 2026) accumulates and refines contextual playbooks through generation, reflection, and curation. Detailed baseline configurations are provided in Appendix B.

Table 2: Ablation studies on history retention, parameter settings, and function selection along the ToolUse → FinQA → Bio stream. $\mathcal{J}_{\text{plas}}$ measures the score change on the current task, while $\mathcal{J}_{\text{stab}}$ measures the mean score change over previously encountered tasks with equal weights. Values are reported in percentage points; higher is better. Stability is undefined at the first stage and denoted by –. Baselines are shown in light beige, default configurations are shaded light blue, and the oracle reference is shown in gray. Bold means with pale gold shading indicate the best result in each column within each ablation group, excluding the baseline and oracle references.

Setting	after ToolUse		after FinQA		after Bio	
	$\mathcal{J}_{\text{plas}}\uparrow$	$\mathcal{J}_{\text{stab}}\uparrow$	$\mathcal{J}_{\text{plas}}\uparrow$	$\mathcal{J}_{\text{stab}}\uparrow$	$\mathcal{J}_{\text{plas}}\uparrow$	$\mathcal{J}_{\text{stab}}\uparrow$
GEPA	+1.07 ± 0.42	–	+30.48 ± 1.73	-0.49 ± 0.28	+4.59 ± 0.86	-11.82 ± 1.24
GRPO	+10.28 ± 0.94	–	+39.46 ± 1.65	-0.40 ± 0.31	+20.58= ± 1.29	-8.32 ± 1.07
History Retention						
Last-20	**+8.60** ± 0.73	–	**+44.58** ± 1.36	+1.49 ± 0.24	**+19.52** ± 1.08	**-0.31** ± 0.19
Last-50	+6.76 ± 0.91	–	+40.19 ± 1.62	+1.48 ± 0.31	+16.04 ± 1.27	-1.33 ± 0.38
Last-20+D_{val}	+7.69 ± 0.68	–	+42.29 ± 1.45	**+1.59** ± 0.27	+17.85 ± 1.16	-0.81 ± 0.26
Tolerance ϵ Settings						
0	+3.56 ± 0.57	–	+31.57 ± 1.84	+0.88 ± 0.32	+11.46 ± 1.39	-3.78 ± 0.61
0.1	**+8.60** ± 0.73	–	**+44.58** ± 1.36	**+1.49** ± 0.24	**+19.52** ± 1.08	**-0.31** ± 0.19
0.2	+7.46 ± 0.82	–	+41.82 ± 1.51	+1.37 ± 0.29	+17.35 ± 1.22	-0.90 ± 0.34
0.5	+5.74 ± 0.76	–	+37.52 ± 1.68	+1.26 ± 0.26	+15.29 ± 1.31	-1.44 ± 0.43
1.0	+4.53 ± 0.64	–	+34.12 ± 1.79	+1.12 ± 0.35	+13.04 ± 1.46	-2.57 ± 0.52
Function Selection						
First	+5.70 ± 0.88	–	+38.40 ± 1.67	+1.07 ± 0.33	+15.43 ± 1.34	-1.61 ± 0.46
Random	+5.25 ± 1.12	–	+37.61 ± 2.04	+1.18 ± 0.41	+15.17 ± 1.63	-1.44 ± 0.57
Embedding-20	+5.84 ± 0.79	–	+38.59 ± 1.58	+1.35 ± 0.28	+15.75 ± 1.25	-1.45 ± 0.39
Embedding-50	+8.34 ± 0.77	–	+44.30 ± 1.42	+1.46 ± 0.26	+19.31 ± 1.14	**-0.29** ± 0.22
LLM-selector	**+8.60** ± 0.73	–	**+44.58** ± 1.36	**+1.49** ± 0.24	**+19.52** ± 1.08	-0.31 ± 0.19
Oracle	+12.06 ± 0.65	–	+50.76 ± 1.21	+1.94 ± 0.23	+24.35 ± 0.97	-0.32 ± 0.18

Implementation Details. For the main experiments and agent continual-learning experiments, we use Qwen3-8B (Qwen Team, 2025a) and GPT-5.6-Luna (OpenAI, 2026) as backbone models and also use these two models for refinement. For ablation studies and mechanism analyses, we use Qwen3-8B as the backbone and GPT-5.6-Luna as the refinement model. For single-task experiments, we fix the backbone to Qwen3-8B and evaluate Qwen3-8B and GPT-5.6-Luna separately as refinement models. All models are configured with low reasoning effort. Unless otherwise specified, we set the historical consistency tolerance to $\epsilon = 0.1$ and retain the most recent $H = 20$ successfully absorbed examples for each function. We use the refinement model with a dedicated routing prompt to select functions. Single-task optimization runs once; other experiments are repeated three times, reporting mean and standard deviation. Further implementation details are provided in Appendix B.

4.2 Main Results

Observation 1: EFRE achieves strong continual-learning performance and exhibits functional reuse and specialization. We evaluate Qwen3-8B and GPT-5.6-Luna as backbone models; † denotes GPT-5.6-Luna refinement, while variants without † use Qwen3-8B. Figure 3 shows that EFRE learns new tasks while largely preserving earlier gains. With Qwen3-8B, EFRE† loses only 1.56 percentage points on FinQA after Bio adaptation, compared with 25.10 for GEPA and 9.43 for ACE, and reaches final ToolUse/FinQA/Bio scores of $66.21/68.26/54.83$. With GPT-5.6-Luna, it retains 73.76% FinQA accuracy, versus 46.24% for GEPA and 47.28% for ACE. Together, these results show a stronger balance between new-task adaptation and retention of previously learned capabilities. Figures 2(a,b) further show that the learned repertoire combines specialization with cross-task reuse. For example, in panel (a), f_1, f_3, and f_5 predominantly serve ToolUse, FinQA, and Bio,

Table 3: Performance comparison across four benchmarks. Both EFRE configurations use Qwen3-8B as the backbone. EFRE and EFRE† use Qwen3-8B and GPT-5.6-Luna as the refinement model, respectively. Aggregate is the unweighted mean across benchmarks, and Improvement is the percentage-point gain over Base. EFRE and EFRE† are shaded light blue and darker blue, respectively. Bold means indicate the best result in each column.

Method	HotpotQA	IFBench	AIME-2025	LiveBench Math	Aggregate	Improvement
Base	42.33	36.90	27.33	48.70	38.82	–
GRPO	43.33	35.88	**38.00**	51.26	42.12	+3.30
ACE	60.00	35.72	31.00	52.76	44.87	+6.06
GEPA	62.33	38.61	32.00	51.95	46.22	+7.41
GEPA+Merge	64.33	28.23	32.00	51.95	44.13	+5.31
EFRE	70.67	39.26	34.33	51.88	49.04	+10.22
EFRE†	**72.33**	**42.38**	36.00	**54.32**	**51.26**	**+12.44**

respectively, whereas f_6 is shared across all three tasks, with utilization rates of 17%, 16%, and 19%. Table 1 quantifies this behavior through cross-task reuse and the number of shared functions, while the example prompts illustrate distinct functional roles. Thus, EFRE does not simply allocate an independent prompt to each task, but develops specialized and reusable functional contexts.

4.3 Ablation Study and Mechanism Analysis

Observation 2: Historical consistency improves continual-learning performance by better balancing plasticity and stability. Table 2 varies history retention and tolerance separately, holding other hyperparameters fixed. We compare the most recent 20 examples (Last-20), 50 examples (Last-50), and 20 examples supplemented with validation data (Last-20+D_{val}), using plasticity and stability defined in Equation. 2 and 3. Last-20 yields the highest plasticity at every stage and the least historical degradation after Bio, showing that more history does not necessarily improve learning. Among the tested tolerances, $\epsilon = 0.1$ achieves the best plasticity and stability at every applicable stage. Its Bio-stage plasticity is +19.52 points, versus +11.46 with strict preservation ($\epsilon = 0$) and +13.04 without a historical performance constraint ($\epsilon = 1$). It exceeds GEPA (+4.59) and approaches GRPO (+20.58), with stability of -0.31 points, versus -11.82 and -8.32, respectively. These results favor a moderate constraint supporting new learning while preserving earlier capabilities.

Observation 3: LLM-based selection enables effective use of the functional repertoire. Table 2 compares selection strategies with other hyperparameters fixed. First always selects the first function, while Random samples uniformly. Embedding-20 and Embedding-50 use Qwen3-Embedding-0.6B fine-tuned on 20 and 50 historical examples per function for retrieval. The LLM selector chooses from the function prompts based on the input, whereas Oracle evaluates all functions and selects the highest-scoring output, providing an upper-bound reference within the repertoire. The LLM selector achieves the highest non-oracle plasticity at every stage and better stability than First and Random. Embedding-50 yields slightly better Bio-stage stability (-0.29 versus -0.31 points), but lower plasticity throughout. Oracle further improves Bio-stage plasticity from +19.52 to +24.35 with comparable stability, indicating room to improve selection within the existing repertoire.

Observation 4: Functional repertoires also benefit single-task optimization. Table 3 shows that EFRE achieves an aggregate score of 49.04 across four benchmarks, compared with 46.22 for GEPA, 44.87 for ACE, and 42.12 for GRPO. Using GPT-5.6-Luna for refinement further improves the aggregate score to 51.26. Figure 2(c) shows that, even within independently optimized single-task settings, utilization is distributed across multiple functions rather than concentrated in a single context. Table 1 reports repertoires of 8 and 12 functions under the two refinement settings, with example functions capturing distinct procedures such as evidence decomposition and equation-based reasoning. These results suggest that functional differentiation is useful beyond continual learning: multiple specialized procedures can emerge within a single task even without task transitions.

Table 4: Minimal agent results across six interactive environments with two backbones. Each backbone uses ICL, GEPA, ACE, or EFRE. Values are mean ± standard deviation; higher is better. EFRE rows are light blue. Bold means indicate the best result within each backbone, including ties.

Method	Blind Spectrum Monitoring	Codebase Adaptation	Cohort Studies	Database Exploration	Exploitable Poker	Sales Prediction
			Qwen3-8B			
+ ICL	12.0 ± 5.6	-32.0 ± 8.4	-28.0 ± 7.9	10.0 ± 5.1	-8.0 ± 6.7	25.0 ± 6.2
+ GEPA	26.0 ± 4.8	-15.0 ± 7.2	-18.0 ± 6.8	22.0 ± 4.6	0.0 ± 5.9	42.0 ± 5.3
+ ACE	2.0 ± 4.4	-10.0 ± 6.9	-24.0 ± 7.3	5.0 ± 4.8	-5.0 ± 5.6	15.0 ± 5.9
+ EFRE	**35.0** ± 4.1	**-5.0** ± 6.2	**-10.0** ± 6.0	**30.0** ± 4.2	**6.0** ± 5.1	**52.0** ± 4.7
			GPT-5.6-Luna			
+ ICL	36.0 ± 4.3	12.0 ± 6.4	0.0 ± 5.8	27.0 ± 4.5	-2.0 ± 5.3	60.0 ± 4.8
+ GEPA	44.0 ± 3.8	18.0 ± 5.7	4.0 ± 5.4	35.0 ± 4.0	8.0 ± 4.8	68.0 ± 4.2
+ ACE	8.0 ± 4.1	**25.0** ± 5.9	-7.0 ± 6.1	12.0 ± 4.4	2.0 ± 5.0	18.0 ± 5.2
+ EFRE	**52.0** ± 3.5	**25.0** ± 5.2	**8.0** ± 4.9	**42.0** ± 3.7	**15.0** ± 4.4	**76.0** ± 3.9

5 Extension: EFRE for Agent Continual Learning

Motivated by EFRE's effectiveness in LLM continual learning, we extend it to a minimal agent system to examine whether its learning mechanism remains effective when training data arise from environmental interactions.

A Minimal Continual-Learning Agent. We adopt the lightweight scaffold from Continual Learning Bench (Asawa et al., 2026) and integrate all methods into the same agent–environment interaction loop. In EFRE, a single frozen LLM performs task execution, function selection, and refinement. Given the current observation, it selects a function and generates actions under its prompt. Interaction feedback guides functional updates, while historical consistency checks govern their acceptance, reuse in another branch, or functional emergence. Adaptation updates the external repertoire while keeping the selector instructions and model parameters fixed.

Evaluation Results. We evaluate six tasks from Continual Learning Bench: Blind Spectrum Monitoring, Codebase Adaptation, Cohort Studies, Database Exploration, Exploitable Poker, and Sales Prediction. Table 4 shows that EFRE achieves the highest mean score on all six tasks with Qwen3-8B. Its unweighted average is 18.0, versus 9.5 for GEPA (Agrawal et al., 2026) and -3.5 for ICL (Brown et al., 2020). Gains include Codebase Adaptation (-5.0 versus GEPA's -15.0) and Sales Prediction (52.0 versus 42.0). With GPT-5.6-Luna (OpenAI, 2026), EFRE averages 36.3, versus 29.5 for GEPA and 22.2 for ICL. It exceeds the strongest baseline on five tasks and matches ACE on Codebase Adaptation. These gains across environments and backbones extend EFRE's benefits to agent continual learning through learning from ongoing environmental interactions.

6 Conclusions

We introduced EFRE, a non-parametric continual learning method that evolves a repertoire of functions represented by prompts while keeping the underlying LLM fixed. Historical consistency checks guide the refinement and reuse of existing functions and the emergence of new ones, allowing the repertoire to adapt as learning tasks arrive. Experiments show that EFRE reduces forgetting and improves overall continual-learning performance, with functional specialization and reuse also benefiting single-task optimization. Ablation studies support the roles of historical consistency and function selection, while an extension to a minimal agent system demonstrates gains in interactive environments. Together, these findings suggest that evolving functional repertoires offer a promising approach to continual learning without model parameter updates. Moreover, EFRE can be naturally extended to agent settings, where its evolving functional repertoire supports continual adaptation from ongoing interactions with dynamic environments.

AI USE STATEMENT

Generative AI tools were used to assist with language drafting and editing based on author-provided research materials and experimental results. The authors reviewed the technical content, verified the reported results against the underlying experiments, and take responsibility for the final text and claims. We did not use generative AI tools to formulate or prove mathematical claims, nor did we use them to generate mathematical proofs.

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, Alex 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. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments. *arXiv preprint arXiv:2606.05661*, 2026. URL https://arxiv.org/abs/2606.05661.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.300. URL https://aclanthology.org/2021.emnlp-main.300/.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Di He, Wenhao Huang, and Tianle Cai. In-place test-time training, 2026. URL https://arxiv.org/abs/2604.06169.

Kehua Feng, Xinyi Shen, Weijie Wang, Xiang Zhuang, Yuqi Tang, Qiang Zhang, and Keyan Ding. SciKnowEval: Evaluating multi-level scientific knowledge of large language models. *arXiv preprint arXiv:2406.09098*, 2024. URL https://arxiv.org/abs/2406.09098.

Anne Harrington, Nayan Saxena, Michael Murphy, Anastasia Borovykh, Zeyu Yun, Sridhar Kamath, Ara Eindra Kyi, Trevor Darrell, Jitendra Malik, and Yutong Bai. When does continual learning require learning. *arXiv preprint arXiv:2607.07847*, 2026. URL https://arxiv.org/abs/2607.07847.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL https://doi.org/10.1073/pnas.1611835114.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://papers.nips.cc/paper/2017/hash/f87522788a2be2d171666752f97ddebb-Abstract.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Mathematical Association of America. MAA invitational competitions, n.d. URL https://maa.org/maa-invitational-competitions/. Section: American Invitational Mathematics Examination (AIME).

OpenAI. GPT-5.6 Luna model. OpenAI API model documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-luna.

Reid Pryzant, Dan Iter, Jerry Li, Yin 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. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1645. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/46499a0622ecf568b72d17b61e45dbd5-Abstract-Datasets_and_Benchmarks_Track.html.

Qwen Team. Qwen3-8B model card. Hugging Face, 2025a. URL https://huggingface.co/Qwen/Qwen3-8B. Accessed September 23, 2026.

Qwen Team. Qwen3-Embedding-0.6B model card. Hugging Face, 2025b. URL https://huggingface.co/Qwen/Qwen3-Embedding-0.6B. Accessed September 23, 2026.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. URL https://arxiv.org/abs/1606.04671.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023. URL https://arxiv.org/abs/2306.05301.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/e4a46394ba5378b3f9a186a5b4c650d1-Abstract-Conference.html.

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. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://www.nature.com/articles/s41586-025-08661-4.

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. URL https://openreview.net/forum?id=eC4ygDs02R.

A Datasets, Benchmarks, and Models

A.1 Datasets and Benchmarks

LLM Continual Learning. We use the ToolUse, FinQA, and SciKE-Bio datasets and evaluation protocols provided by (Harrington et al., 2026), following the sequence ToolUse → FinQA → SciKE-Bio.

- **ToolUse.** ToolUse evaluates tool selection and argument construction from natural-language requests. It is derived from ToolAlpaca, which contains 3,938 simulated tool-use instances covering over 400 APIs across 50 categories (Tang et al., 2023). Each input provides a request and relevant tool information, and the model produces the corresponding tool calls. We use it to study the acquisition and retention of tool-use capabilities.
- **FinQA.** FinQA is a financial question-answering dataset containing expert-annotated questions over financial reports, together with numerical answers and reasoning programs (Chen et al., 2021). Questions require integrating evidence from text and tables and performing calculations involving quantities such as percentages, changes, and ratios. We use it to study continual learning of financial numerical reasoning.
- **SciKE-Bio.** SciKE-Bio is a multiple-choice biology subset derived from SciKnowEval, a benchmark covering scientific knowledge in biology, chemistry, physics, and materials science (Feng et al., 2024). As the final task in the sequence, it allows us to examine scientific knowledge acquisition and its effects on previously learned capabilities.

Single-Task Optimization. We use four benchmarks to assess whether EFRE also improves learning within individual tasks.

- **HotpotQA.** HotpotQA is a Wikipedia-based multi-hop question-answering benchmark with annotated answers and supporting facts (Yang et al., 2018). Its questions require combining evidence across documents, including identifying connections between entities and comparing their attributes. We use it to assess multi-hop reasoning and evidence-based question answering.
- **IFBench.** IFBench evaluates instruction following through automatically verifiable constraints, including output formatting, lexical requirements, and combinations of instructions (Pyatkin et al., 2025). It emphasizes generalization to constraints beyond those encountered during training. We use it to assess the model's ability to follow diverse instructions.
- **AIME-2025.** AIME-2025 contains the 30 problems from the two 2025 American Invitational Mathematics Examinations (Mathematical Association of America, n.d.). These problems require competition-level mathematical reasoning and have integer answers. We use it to assess mathematical problem solving.
- **LiveBench-Math.** LiveBench-Math is the mathematical subset of LiveBench, a regularly updated benchmark with automatically checkable answers (White et al., 2025). We use it to assess mathematical reasoning on an additional set of problems.

Agent Continual Learning.

- **Continual Learning Bench.** Continual Learning Bench evaluates continual adaptation through sequential interactions in stateful environments (Asawa et al., 2026). Its tasks require agents to accumulate information, learn recurring environmental structure, and adapt to changes across interactions. We evaluate six environments: Blind Spectrum Monitoring, Codebase Adaptation, Cohort Studies, Database Exploration, Exploitable Poker, and Sales Prediction. Together, they test whether EFRE supports continual learning from agent–environment interactions.

A.2 Models

- **Qwen3-Embedding-0.6B.** Qwen3-Embedding-0.6B is an instruction-aware text embedding model with approximately 0.6 billion parameters, a 32K-token context window, and

Table 5: Default EFRE hyperparameters. Model choices correspond to the experimental configurations reported in the main text.

Hyperparameter	Setting
Micro-batch size	3
History capacity H	20 examples per function
Historical consistency tolerance ϵ	0.1
Maximum number of functions	50
Backbone model	Qwen3-8B, GPT-5.6-Luna
Refinement model	Qwen3-8B, GPT-5.6-Luna
Backbone reasoning effort	Low
Refinement reasoning effort	Low
Embedding model	Qwen3-Embedding-0.6B

configurable embedding dimensions from 32 to 1,024 (Qwen Team, 2025b). We use it for the Embedding-20 and Embedding-50 function-selection baselines, fine-tuning on 20 and 50 historical examples per function, respectively.

- **Qwen3-8B.** Qwen3-8B is an open-weight language model with 8.2 billion parameters and a native context window of 32,768 tokens, supporting both thinking and non-thinking modes (Qwen Team, 2025a). We use it as a backbone, refinement model, and function selector in the corresponding experimental configurations. Its parameters remain fixed under EFRE.
- **GPT-5.6-Luna.** GPT-5.6-Luna is an API-served language model supporting reasoning, tool calling, and structured outputs (OpenAI, 2026). We use it as a backbone, refinement model, and function selector, with reasoning effort set to Low. EFRE† uses GPT-5.6-Luna for refinement, whereas EFRE without † uses Qwen3-8B.

B Implementation Details

B.1 Hyperparameters

Table 5 summarizes the default EFRE configuration. Unless otherwise specified, we use a micro-batch size of 3, retain the 20 most recent successfully absorbed examples per function, and set the historical consistency tolerance to 0.1. The maximum repertoire size is 50. The backbone and refinement models follow the configurations reported in each experiment. All experiments are repeated three times.

Table 6 summarizes optimizer and batch-size settings for the comparison methods. Parameter-updating methods use AdamW (Loshchilov & Hutter, 2019) with a batch size of 16; GEPA and ACE use batches of 3 examples for textual updates. The Embedding-20 and Embedding-50 ablations restrict embedding-model training to 20 and 50 historical examples per function, respectively.

B.2 Dataset Setup

For LLM continual learning, we follow the dataset setup of (Harrington et al., 2026), with tasks arriving in the order ToolUse $\rightarrow$ FinQA $\rightarrow$ SciKE-Bio. Each single-task benchmark is optimized independently. Training, validation, and evaluation splits are disjoint and shared across methods. Training data provide optimization examples and feedback, validation data support update selection, and evaluation data measure performance. Function histories are drawn from training data; the Last-20+D_{val} ablation additionally uses validation data for consistency checks. For AIME-2025, training and validation questions come from 2022–2024, while evaluation uses the 30 questions from 2025. Agent continual learning follows ordered interactions in Continual Learning Bench rather than a static three-way split. Updates use only information available from completed interactions.

Table 6: Optimizer and batch-size settings for comparison methods. A dash indicates that the setting is not applicable.

Method	Optimizer	Batch size
SFT	AdamW	16
SDFT	AdamW	16
SDPO	AdamW	16
GRPO	AdamW	16
In-Place TTT	AdamW	16
GEPA	–	3
ACE	–	3
ICL	–	–

B.3 Evaluation Protocols

Task Scoring. We evaluate each benchmark on its held-out evaluation set D_{eval} using the official scoring criteria. Evaluation examples are excluded from training, prompt refinement, and update selection. The same evaluation sets and scoring procedures are used across methods.

Evaluation Matrix. Let T denote the number of tasks in the continual-learning sequence, S_0 the initial learner state, and S_t the state after learning task t. For EFRE, S_t includes the functional repertoire and selector, while the backbone parameters remain fixed. For parameter-updating baselines, evaluation uses the model checkpoint obtained after the corresponding stage.

We evaluate every task before learning begins and after each learning stage, forming the performance matrix

$$A_{t,i} = R(S_t; D_{\text{eval},i}),\ 0 \le t \le T, \quad 1 \le i \le T, \tag{9}$$

where $A_{t,i}$ is the score on task i after stage t. Each row therefore describes one learner state across all tasks, and each column tracks performance on one task throughout learning. Row $t = 0$ contains the initial scores before any adaptation.

Evaluation leaves the learner unchanged. For EFRE, each input is routed by the function selector, and the selected function is used to generate the response. Reported scores thus evaluate the complete system, including function selection. Oracle selection is used only in the explicitly designated oracle ablation. For the LLM continual-learning experiments, we express $A_{t,i}$ on a 0–100 scale; differences between these scores are reported in percentage points.

Plasticity. Plasticity measures how much learning at stage t improves performance on the current task:

$$\mathcal{J}^{(t)}_{\text{plas}} = A_{t,t} - A_{t-1,t}. \tag{10}$$

Here, $A_{t-1,t}$ is the score on task t immediately before learning it, and $A_{t,t}$ is the score immediately afterward. The comparison therefore isolates the performance change during the current stage, including when earlier learning has already affected the current task. A positive value indicates improvement, zero indicates no change, and a negative value indicates degradation.

For the ToolUse $\rightarrow$ FinQA $\rightarrow$ SciKE-Bio sequence, the three plasticity values are

$$\begin{aligned} \mathcal{J}^{(1)}_{\text{plas}} &= A_{1,1} - A_{0,1}, \\ \mathcal{J}^{(2)}_{\text{plas}} &= A_{2,2} - A_{1,2}, \\ \mathcal{J}^{(3)}_{\text{plas}} &= A_{3,3} - A_{2,3}. \end{aligned} \tag{11}$$

For example, FinQA plasticity compares its score after FinQA adaptation with its score after ToolUse adaptation, rather than with the initial model's FinQA score.

Stability. Stability measures how learning at stage t changes performance on previously learned tasks:

$$\mathcal{J}^{(t)}_{\text{stab}} = \frac{1}{t-1} \sum_{i=1}^{t-1} \left(A_{t,i} - A_{t-1,i}\right),\ t > 1. \tag{12}$$

For each previous task, we compare its scores immediately before and after the current stage, then average these changes with equal weight across tasks. This weighting prevents tasks with larger evaluation sets from dominating the measure.

A negative value indicates average forgetting, zero indicates no net change in previous-task performance, and a positive value indicates average improvement. Because the measure averages across tasks, gains on one task can offset losses on another. We therefore report individual task scores alongside the aggregate stability measure. Stability is undefined at the first stage, when no previous tasks exist, and is denoted by "–".

For the three-task sequence,

$$\begin{aligned} \mathcal{J}^{(2)}_{\text{stab}} &= A_{2,1} - A_{1,1}, \\ \mathcal{J}^{(3)}_{\text{stab}} &= \frac{1}{2}\left[(A_{3,1} - A_{2,1}) + (A_{3,2} - A_{2,2})\right]. \end{aligned} \tag{13}$$

Thus, stability after FinQA measures the change on ToolUse, while stability after SciKE-Bio averages the changes on ToolUse and FinQA. This definition measures the effect of the current learning stage. It differs from cumulative backward transfer (Lopez-Paz & Ranzato, 2017), which compares each previous task's current score with its score immediately after that task was originally learned.

Aggregation and Uncertainty. Final continual-learning performance is the unweighted mean of all task scores after the last stage:

$$\overline{A}_{\text{final}} = \frac{1}{T}\sum_{i=1}^{T} A_{T,i}. \tag{14}$$

For single-task optimization, we similarly report the unweighted mean of the four benchmark scores. Each benchmark contributes equally to the aggregate, regardless of its evaluation-set size.

All experiments are repeated three times. We first compute task aggregates, plasticity, and stability separately within each run, and then calculate their means and sample standard deviations across runs. For any reported quantity z, let $z^{(r)}$ denote its value in run r. We compute

$$\bar{z} = \frac{1}{3}\sum_{r=1}^{3} z^{(r)}, \;\; s_z = \sqrt{\frac{1}{2}\sum_{r=1}^{3}\left(z^{(r)} - \bar{z}\right)^2}. \tag{15}$$

Results with uncertainty estimates are reported as $\bar{z} \pm s_z$. For plasticity and stability, the before-and-after scores are paired within the same run before computing the standard deviation. Their uncertainty cannot be recovered from the separate checkpoint standard deviations alone, because the scores within a run may be correlated. Tables calculated from aggregate checkpoint scores therefore report means only. Small rounding differences may arise when calculations use displayed scores rather than the original unrounded values.

Agent Evaluation. We evaluate agents through the ordered interaction schedule of each environment in Continual Learning Bench (Asawa et al., 2026). For environment e, let $r_{e,j}$ be the native reward on interaction instance j, and let N_e be the number of evaluated instances. The environment-level mean reward is

$$\bar{r}_e = \frac{1}{N_e}\sum_{j=1}^{N_e} r_{e,j}. \tag{16}$$

Native reward definitions differ across environments. We therefore report the benchmark's normalized reward:

$$\widehat{r}_e = 100\,\frac{\bar{r}_e - r_e^{\text{ref}}}{r_e^{\max} - r_e^{\text{ref}}}, \tag{17}$$

where r_e^{ref} is the benchmark's fixed reference reward and $r_e^{\max}$ is its designated upper reference value, with $r_e^{\max} > r_e^{\text{ref}}$. The same normalization constants are used for all methods evaluated in that environment. A normalized reward of zero matches the reference, 100 matches the upper reference value, and a negative value indicates performance below the reference. These scores are normalized rewards, not accuracy percentages.

Table 7: Full continual-learning results. Each stage reports ToolUse, FinQA, and SciKE-Bio accuracy (%); Bio denotes SciKE-Bio. Final Avg. is the unweighted mean after Bio. Bold values indicate the highest score in each column within each backbone, including ties. Light and dark blue rows denote EFRE and EFRE†, respectively.

Method	Base			After ToolUse			After FinQA			After Bio			Final Avg.
	ToolUse	FinQA	Bio	ToolUse	FinQA	Bio	ToolUse	FinQA	Bio	ToolUse	FinQA	Bio	
Qwen3-8B													
SFT	**58.17**	24.31	29.84	60.26	21.73	26.18	56.79	29.24	28.36	57.82	10.27	36.74	34.94
SDFT	57.28	24.16	30.22	63.76	28.34	34.81	64.21	69.74	29.36	62.18	**68.83**	43.27	58.09
SDPO	58.11	23.79	30.26	64.83	**30.21**	35.17	65.24	62.78	33.69	64.72	58.31	53.18	58.74
GRPO	57.86	**26.23**	**33.78**	**68.14**	26.37	**35.82**	**67.74**	65.83	29.28	51.22	65.71	49.86	55.60
GEPA	55.24	23.17	31.84	56.31	22.76	32.19	55.82	53.24	31.73	57.28	28.14	36.32	40.58
ACE	54.16	22.83	32.27	52.31	23.18	31.76	51.79	44.26	32.14	53.24	34.83	38.29	42.12
In-Place TTT	57.81	24.27	29.76	52.23	24.82	30.14	45.18	23.76	27.82	31.24	22.17	34.26	29.22
EFRE	55.21	23.14	31.86	60.32	26.18	31.27	62.24	65.83	35.16	61.18	63.76	51.32	58.75
EFRE†	55.18	23.21	31.82	63.78	25.24	32.16	65.27	**69.82**	**35.31**	**66.21**	68.26	**54.83**	**63.10**
GPT-5.6-Luna													
GEPA	66.24	29.18	37.83	71.26	30.17	39.24	70.82	68.31	41.16	69.76	46.24	49.83	55.28
ACE	**67.18**	**29.24**	38.16	63.82	30.26	38.81	64.27	55.78	40.23	65.16	47.28	48.19	53.54
EFRE†	66.27	29.16	**38.23**	**75.24**	**33.18**	**41.27**	**76.31**	**77.82**	**44.24**	**74.83**	**73.76**	**58.29**	**68.96**

Table 4 reports normalized reward for each environment. The overall agent score assigns equal weight to the six environments:

$$\overline{r}_{\text{agent}} = \frac{1}{6}\sum_{e=1}^{6}\widehat{r}_e. \tag{18}$$

Normalization and aggregation are performed within each run before summarizing results across the three runs.

Normalized reward differs from normalized gain, which compares an adapting agent with its own stateless counterpart:

$$\widehat{g}_e = 100\,\frac{\bar{r}_e^{\text{stateful}} - \bar{r}_e^{\text{stateless}}}{r_e^{\max} - \bar{r}_e^{\text{stateless}}}. \tag{19}$$

Normalized gain requires a separate stateless evaluation and a positive denominator. Our agent results report normalized reward; normalized gain is not used. Neither quantity is the stage-local stability measure $\mathcal{J}_{\text{stab}}$ defined for the sequential LLM tasks above.

C Additional Results

C.1 Full Results

Tables 7 and 8 report stage-wise performance for all 12 configurations. EFRE† uses GPT-5.6-Luna for refinement, whereas unmarked EFRE uses Qwen3-8B. With Qwen3-8B as the backbone, EFRE† achieves a final average of 63.10, exceeding GRPO and GEPA by 7.50 and 22.52 points, respectively. Its Bio-stage plasticity is +19.52, close to GRPO's +20.58, while its stability is substantially higher (-0.31 versus -8.32). Using Qwen3-8B for refinement also yields a final average of 58.75 with Bio-stage stability of -1.57. With GPT-5.6-Luna as the backbone, EFRE† reaches 68.96, compared with 55.28 for GEPA and 53.54 for ACE. These results show that EFRE combines effective adaptation with stronger retention across both backbones.

C.2 Cross-Model Analysis of Functional Repertoires

We consider transfer between Qwen3-8B and GPT-5.6-Luna, using GPT-5.6-Luna as the refinement model throughout. The source model is the backbone used to learn a repertoire; the target model is

Table 8: Stage-local plasticity and stability calculated from Table 7, in percentage points (higher is better). Bold values indicate the highest value in each column within each backbone, including ties. First-stage stability is undefined and shown as a dash. Light and dark blue rows denote EFRE and EFRE†, respectively.

Method	**After ToolUse**		**After FinQA**		**After Bio**	
	$\mathcal{J}_{\text{plas}} \uparrow$	$\mathcal{J}_{\text{stab}} \uparrow$	$\mathcal{J}_{\text{plas}} \uparrow$	$\mathcal{J}_{\text{stab}} \uparrow$	$\mathcal{J}_{\text{plas}} \uparrow$	$\mathcal{J}_{\text{stab}} \uparrow$
Qwen3-8B						
SFT	+2.09	—	+7.51	-3.47	+8.38	-8.97
SDFT	+6.48	—	+41.40	+0.45	+13.91	-1.47
SDPO	+6.72	—	+32.57	+0.41	+19.49	-2.50
GRPO	**+10.28**	—	+39.46	-0.40	**+20.58**	-8.32
GEPA	+1.07	—	+30.48	-0.49	+4.59	-11.82
ACE	-1.85	—	+21.08	-0.52	+6.15	-3.99
In-Place TTT	-5.58	—	-1.06	-7.05	+6.44	-7.77
EFRE	+5.11	—	+39.65	**+1.92**	+16.16	-1.57
EFRE†	+8.60	—	**+44.58**	+1.49	+19.52	**-0.31**
GPT-5.6-Luna						
GEPA	+5.02	—	+38.14	-0.44	+8.67	-11.57
ACE	-3.36	—	+25.52	+0.45	+7.96	-3.81
EFRE†	**+8.97**	—	**+44.64**	**+1.07**	**+14.05**	**-2.77**

Table 9: Cross-model evaluation of functional repertoires. The arrow denotes source-to-target backbone transfer; Q and L abbreviate Qwen3-8B and GPT-5.6-Luna. All configurations use GPT-5.6-Luna refinement. Each stage reports ToolUse, FinQA, and SciKE-Bio accuracy (%). Final Avg. averages the three scores after Bio. Bold values mark column maxima, including ties.

Source → Target	**Base**			**After ToolUse**			**After FinQA**			**After Bio**			**Final Avg.**
	ToolUse	FinQA	Bio	ToolUse	FinQA	Bio	ToolUse	FinQA	Bio	ToolUse	FinQA	Bio	
Q → Q	55.18	23.21	31.82	63.78	25.24	32.16	65.27	69.82	35.31	66.21	68.26	54.83	63.10
Q → L	**66.27**	**29.16**	**38.23**	72.86	31.94	40.58	74.12	75.46	43.17	72.38	71.84	57.17	67.13
L → Q	55.18	23.21	31.82	65.42	27.06	33.18	67.83	72.14	37.62	68.47	70.31	56.28	65.02
L → L	**66.27**	**29.16**	**38.23**	**75.24**	**33.18**	**41.27**	**76.31**	**77.82**	**44.24**	**74.83**	**73.76**	**58.29**	**68.96**

the backbone used to execute it. At each stage, the source repertoire is evaluated with the target backbone without further function refinement. The same-model configurations provide reference scores. The base columns correspond to the target model before adaptation. Table 9 shows that functional repertoires remain useful when transferred between backbones, while performance also depends on the model executing them. Transferring the GPT-5.6-Luna repertoire to Qwen3-8B raises its final average from 63.10 to 65.02 (+1.92 points), with gains on all three tasks. Conversely, executing the Qwen3-8B repertoire with GPT-5.6-Luna achieves 67.13, exceeding its original Qwen3-8B execution by 4.03 points and the reverse-transfer configuration by 2.11 points. The same repertoire performs better with GPT-5.6-Luna in both comparisons: by 4.03 points for the Qwen3-8B repertoire and 3.94 points for the GPT-5.6-Luna repertoire. Nevertheless, GPT-5.6-Luna achieves its best result with its own repertoire (68.96), exceeding the transferred Qwen3-8B repertoire by 1.83 points. Together, these results indicate that backbone capability strongly influences performance, while learned functional contexts retain utility across models.

C.3 Training Cost

We report adaptation costs over the complete three-stage continual-learning sequence in Table 10. The accounting includes rollout generation, prompt refinement, candidate evaluation, historical con-

Table 10: Measured adaptation workloads and elapsed times for 12 configurations over a complete three-stage run. API output cost is calculated at $1.20 per million output tokens, and local GPU cost at $2.00 per GPU-hour. Accounted cost includes API output charges and local GPU cost but excludes API input charges. EFRE† uses GPT-5.6-Luna for refinement. Blue rows identify EFRE variants.

Method	API calls	Output tokens (M)	API output cost (USD)	GPU hours	Accounted cost (USD)	Wall time (h)
			Qwen3-8B			
SFT	0	0.00	0.00	18.2	36.40	4.8
SDFT	0	0.00	0.00	40.6	81.20	10.5
SDPO	0	0.00	0.00	50.4	100.80	13.0
GRPO	0	0.00	0.00	75.6	151.20	19.5
GEPA	1,400	9.60	11.52	6.0	23.52	3.7
ACE	2,300	17.40	20.88	12.8	46.48	7.1
In-Place TTT	0	0.00	0.00	32.2	64.40	8.4
EFRE	0	0.00	0.00	6.8	13.60	3.9
EFRE†	1,050	7.20	8.64	5.0	18.64	4.9
			GPT-5.6-Luna			
GEPA	2,100	15.00	18.00	0.0	18.00	3.8
ACE	3,400	26.40	31.68	0.0	31.68	6.5
EFRE†	2,650	19.20	23.04	0.0	23.04	4.7

sistency checks, function reuse attempts, and parameter updates, as applicable to each method. Final held-out evaluation is excluded.

Local computation uses up to four NVIDIA A100-80GB GPUs. Parameter-updating methods incur local computation for optimization and, where applicable, rollout generation and teacher inference. Prompt-based methods incur computation for candidate generation, refinement, and evaluation. EFRE additionally performs historical consistency checks and function reuse or emergence decisions. With Qwen3-8B as the backbone, GEPA and ACE use hosted refinement, default EFRE operates locally, and EFRE† uses GPT-5.6-Luna for refinement. With GPT-5.6-Luna as the backbone, both task execution and refinement use hosted APIs.

API call counts, output-token counts, GPU-hours, and wall-clock times are measured over the complete adaptation sequence. We calculate API output cost at $1.20 per million output tokens and local GPU cost at $2.00 per GPU-hour. The reported accounted cost is the sum of these two components:

$$C_{\text{accounted}} = 1.20\, N_{\text{out}}^{(\text{M})} + 2.00\, H_{\text{GPU}},$$

where $N_{\text{out}}^{(\text{M})}$ is the API output-token count in millions and H_{GPU} is the aggregate local GPU usage in GPU-hours. API input-token charges are excluded, so accounted cost does not represent the complete adaptation expenditure. GPU-hours aggregate allocated GPU time across devices, whereas wall-clock time includes execution, API waiting, and orchestration. Hosted computation is not counted as local GPU usage.

With Qwen3-8B, default EFRE uses 6.8 GPU-hours and has the lowest accounted cost among the evaluated configurations at $13.60. Its local GPU usage is slightly higher than GEPA's 6.0 GPU-hours, but avoiding hosted refinement reduces its accounted cost relative to GEPA's $23.52. Their wall-clock times are 3.9 and 3.7 hours, respectively. EFRE† reduces local GPU usage to 5.0 GPU-hours while incurring $8.64 in API output charges, resulting in an accounted cost of $18.64 and a wall-clock time of 4.9 hours. Both EFRE variants use fewer local GPU-hours than SFT, SDFT, SDPO, GRPO, and In-Place TTT in this setting. These comparisons reflect both the adaptation procedures and their refinement backends.

With GPT-5.6-Luna as the backbone, no local GPU computation is recorded, and accounted cost consists entirely of API output charges. EFRE† uses 19.20 million output tokens and costs $23.04, compared with $18.00 for GEPA and $31.68 for ACE. Its wall-clock time of 4.7 hours also falls

between GEPA's 3.8 hours and ACE's 6.5 hours. Thus, EFRE† incurs lower output-token cost and shorter elapsed time than ACE, while GEPA is less costly and faster in this setting. All monetary comparisons use the stated rates and exclude API input-token charges.

D EFRE DETAILS

This section describes the complete EFRE update procedure, explains how historical consistency governs reuse and emergence, and specifies how functions are selected at inference.

D.1 ALGORITHM WORKFLOW

Stages and Update Batches. Algorithm 1 describes learning across task stages, while Algorithm 2 describes the update performed on one training batch. At stage t, EFRE processes micro-batches drawn from $D_{\mathrm{train},t}$, using a default micro-batch size of 3. The repertoire persists across batches and stages. We denote an individual update batch by B; if a stage contains only one update batch, then $B = D_{\mathrm{train},t}$. Stage indices therefore identify task transitions, rather than individual prompt revisions.

The repertoire initially contains one function represented by an initial prompt, with an empty history. This is the current function for the first batch. Subsequently, EFRE starts each batch from the branch accepted by the most recent successful update, including a branch accepted through reuse or emergence. If an update is rejected, the current branch remains unchanged. The entire micro-batch is processed by this current function; the inference selector is not used to assign its individual examples to different branches.

Candidate Evaluation. For compactness, the pseudocode writes $V = D_{\mathrm{val},t}$ and $H_k = D_{\mathrm{history},k}$, and uses $\mathcal{D}$ for the collection of per-function histories. The scalar H denotes history capacity and is distinct from the dataset H_k. All scores $R(f; D)$ in a functional update are obtained by activating function f directly, together with the fixed shared base instructions. Function selection is bypassed for these local comparisons so that candidate and original prompts are evaluated under the same conditions.

The frozen backbone first runs on B with the current function. The refinement model receives the function prompt, batch, and execution feedback, and produces a candidate f'_k. The candidate must improve both training and validation scores:

$$R(f'_k; B) > R(f_k; B),\ R(f'_k; V) > R(f_k; V). \tag{20}$$

Both comparisons use the original function f_k as the baseline. A tie on either dataset does not satisfy this improvement rule. If either comparison fails, the candidate is discarded and processing of this batch ends without searching other functions.

Historical Consistency and Update Outcomes. A candidate satisfying both improvement conditions is evaluated on the current function's history. It passes if

$$R(f'_k; H_k) \geq (1 - \epsilon) R(f_k; H_k), \tag{21}$$

where scores are nonnegative and $\epsilon \in [0, 1]$ is the maximum permitted proportional decrease. For example, $\epsilon = 0.1$ permits a decrease of at most 10% of the original historical score. When H_k is empty, this check passes without computing a score on an empty dataset.

A passing candidate replaces the current function in place. Only a candidate that improves current performance but fails historical consistency triggers traversal of other functions. The search accepts the first compatible revision. If none is found and capacity remains, EFRE proposes a new function, which must also pass the training and validation improvement checks. Thus, other branches are evaluated only when the current update encounters historical incompatibility. Algorithm 2 returns the updated repertoire, its histories, and the index of the function that absorbed the batch. An empty index indicates that no update was accepted.

The maximum repertoire size is 50. Existing functions can still be refined or reused at this limit. If the search finds no compatible update when the repertoire is full, the update is discarded and the repertoire remains unchanged.

Algorithm 1 EFRE across a sequence of learning stages

Require: Frozen Φ, refinement model M_{refine}; initial function f_1; fixed textual selector σ; stage datasets; history capacity H; tolerance ϵ; capacity $K_{\max} = 50$

$\mathcal{F} \leftarrow \{f_1\}$; $H_1 \leftarrow \varnothing$; $\mathcal{D} \leftarrow \{H_1\}$; $k \leftarrow 1$
Evaluate the initial system to obtain $A_{0,i}$ for every task i
for $t = 1, \ldots, T$ **do**
  for each update batch B from $D_{\text{train},t}$ **do**
    $(\mathcal{F}, \mathcal{D}, j) \leftarrow \text{UPDATEFUNCTION}(\mathcal{F}, \mathcal{D}, k, B, D_{\text{val},t}, H, \epsilon, K_{\max})$
    if $j \neq \varnothing$ **then**
      $k \leftarrow j$ ▷ Continue from the accepted branch
    end if
  end for
  Save S_t containing the repertoire, histories, shared instructions, and selector
  Set $A_{t,i} \leftarrow R(S_t; D_{\text{eval},i})$ for every task i, without state changes
end for
return S_T and the evaluation matrix A

Algorithm 2 One EFRE update: refinement, conditional reuse, and emergence

Require: Access to frozen Φ, M_{refine}, and shared base instructions

function UPDATEFUNCTION($\mathcal{F}, \mathcal{D}, k, B, V, H, \epsilon, K_{\max}$)
  Run Φ with f_k on B and obtain feedback $\mathcal{Q}_k$
  $f'_k \leftarrow M_{\text{refine}}(f_k, B, \mathcal{Q}_k)$
  if either improvement condition in Equation 20 fails **then**
    return $(\mathcal{F}, \mathcal{D}, \varnothing)$ ▷ No search or emergence
  end if
  if H_k is empty or Equation 21 holds **then**
    Replace f_k by f'_k at the same position
    Append successfully absorbed training examples to H_k; keep the newest H
    return $(\mathcal{F}, \mathcal{D}, k)$
  end if
  Record the current failed proposal and its feedback
  for each $j \neq k$ in the stored repertoire order **do**
    Run Φ with f_j on B and obtain $\mathcal{Q}_j$
    $f'_j \leftarrow M_{\text{refine}}(f_j, B, \mathcal{Q}_j)$
    if both improvement conditions hold for j **then**
      if H_j is empty or historical consistency holds for j **then**
        Replace f_j by f'_j at the same position
        Append successfully absorbed examples to H_j; keep the newest H
        return $(\mathcal{F}, \mathcal{D}, j)$ ▷ First compatible branch
      end if
    end if
    Record the failed proposal and evaluation feedback
  end for
  if $|\mathcal{F}| \geq K_{\max}$ **then**
    return $(\mathcal{F}, \mathcal{D}, \varnothing)$
  end if
  Initialize f_{new} with M_{refine} using B and failed-attempt feedback
  if $R(f_{\text{new}}; B) \leq R(f_k; B)$ or $R(f_{\text{new}}; V) \leq R(f_k; V)$ **then**
    return $(\mathcal{F}, \mathcal{D}, \varnothing)$
  end if
  Evaluate f_{new} to identify successfully handled examples in B
  Append f_{new} to $\mathcal{F}$; add its history of at most H successful examples to $\mathcal{D}$
  return $(\mathcal{F}, \mathcal{D}, |\mathcal{F}|)$
end function

D.2 REUSE AND EMERGENCE

Searching for a Compatible Function. After a history-incompatible update, EFRE visits the remaining functions in their stored order, excluding the current function. For each alternative f_j, it runs the incoming batch using that function, obtains function-specific feedback, and generates a new candidate f'_j. The candidate is compared with f_j on B and V, and with f_j on its own history H_j. It

is not compared with the original branch f_k. This procedure tests whether the batch can improve an existing function without excessive loss of that function's prior behavior.

The search stops at the first candidate satisfying all three conditions. EFRE replaces only that function and updates its history; the original function and every other branch remain unchanged. There is no maximization over all candidates, and functions after the accepted branch are not evaluated. A rejected candidate from the original branch is not copied into another branch: each alternative receives its own revision. This allows the same function to incorporate compatible learning signals from different tasks.

Creating a New Function. If every alternative candidate fails acceptance and repertoire capacity remains available, the refinement model initializes a new function from the incoming batch and feedback collected during the failed attempts. Before admission, it must improve both training and validation performance relative to the previous prompt, taken here as the current function f_k before this batch update:

$$R(f_{\text{new}}; B) > R(f_k; B),\ R(f_{\text{new}}; V) > R(f_k; V). \tag{22}$$

If either condition fails, the new candidate is discarded. Otherwise, it is appended to the repertoire, its history is initialized with successfully handled examples, and it becomes the current branch for the next batch. It has no previous history to preserve. This automatic expansion is termed functional emergence. It is driven by the update process rather than a predefined assignment of one function to each task.

Failure of the search means that none of the generated candidates met the acceptance conditions; it does not establish that every possible revision would be incompatible. Refinement, reuse, and emergence together allow both function content and repertoire size to evolve as learning proceeds.

Maintaining Function Histories. Each history stores recent training examples successfully absorbed by its function. Following an accepted update, EFRE appends the newly absorbed examples and removes the oldest entries if the history exceeds H examples. Rejected candidates change neither the stored prompt nor its history. Histories therefore remain local to individual functions and contain no held-out evaluation examples. The training data contain no duplicate examples.

Historical consistency constrains the aggregate score on this finite buffer. It does not require unchanged performance on every stored example, nor does it directly evaluate every past task. Moreover, each check compares a candidate with its immediate predecessor, so permitted losses can accumulate over successive updates. System-level retention is therefore measured separately on the held-out task sets using $\mathcal{J}_{\text{stab}}$.

Feedback in Interactive Environments. In the agent setting, EFRE saves historical interactions and replays them when evaluating functional updates. Replay provides the basis for comparing a candidate with the existing function on previously encountered interactions. Feedback from current interactions supports refinement, while replay assesses retention of historical behavior. These checks use past interactions rather than future evaluation instances.

D.3 Function Selector and Inference

Selecting and Executing a Function. At inference time, the textual selector receives an input x and the function prompts in $\mathcal{F}_t$, and returns a function index. The frozen backbone then generates the answer using the selected function and the shared base instructions:

$$k(x) = \sigma(x; \mathcal{F}_t),\ \hat{y} = \Phi(x; p_{\text{base}}, f_{k(x)}). \tag{23}$$

The selector does not receive task labels, reference answers, or evaluation scores. Both the repertoire and selector remain fixed during held-out evaluation.

The selector uses a fixed instruction prompt and is not rewritten during continual learning. Its available function prompts change as the repertoire evolves, allowing it to select newly created or revised functions without updating its own instructions. Thus, training proceeds from the most recently updated branch, whereas inference selects a function separately for each input. All functional adaptation remains in the external textual state; the backbone parameters remain fixed.

E ILLUSTRATIVE EXAMPLES AND PROMPTS

Selection Prompt

Instruction: Select the function best suited to the current query. Match the query's requirements to each function's description and scope rather than relying only on shared keywords. Do not answer the query or modify the repertoire.

Inputs

Function repertoire: A collection of function IDs and descriptions specifying the operations each function supports.
`f1` — **Tool execution:** Select a tool, validate its arguments, and inspect the result.
`f2` — **Evidence retrieval:** Retrieve and verify facts needed to answer a question.
`f3` — **Numerical reasoning:** Extract quantities, perform calculations, and verify results.
`f4` — **Constraint checking:** Check an answer against explicit output requirements.
Current query: The question or immediate task requiring a function.
Example: Revenue increased from $80 million in 2023 to $100 million in 2024. What was the percentage increase?

Output

Return only the selected function's ID.
Example: `f3`

Figure 4: Prompt for selecting a function from the function library.

Refinement Prompt

Instruction: Revise the old prompt using the feedback and the three queries. Address the instruction-level causes of the reported failures while preserving useful instructions and the function's scope. Prefer reusable guidance over query-specific answers. Keep the revised prompt concise and internally consistent.

Inputs

Old prompt: The existing textual context of the function being refined.
Example: Extract the relevant values, calculate their difference, and report the result to two decimal places.
Feedback: Evaluation results describing errors or missing instructions observed when applying the old prompt to the queries.
Example: For Query 1, the response reported the absolute increase instead of the percentage increase. For Query 2, it confused percentage points with relative change. For Query 3, it subtracted values without converting their units.
Queries: Three questions used to identify the required improvements.
1. Revenue rose from $80 million to $100 million. What was the percentage increase?
2. A margin increased from 12% to 15%. How many percentage points did it increase?
3. Revenue was $1.2 billion and costs were $950 million. What was the difference in millions of dollars?

Output

Return only the complete revised prompt, without commentary or answers to the queries.

Figure 5: Prompt for refining a functional context using feedback and three queries.

Optimized Prompt from GEPA

Base prompt: Answer the question accurately using the available evidence and tools. Explain the reasoning when needed and follow the requested output format.

After optimization
You are an assistant for tool use, financial analysis, biological questions, multi-hop retrieval, and mathematical reasoning. Apply the following instructions when interpreting requests, selecting tools, and constructing answers. These instructions summarize adjustments made after earlier task failures. When a question resembles a previous example, begin with the procedure that succeeded on that example.

Financial questions
For revenue questions, inspect the last table first and use its final numerical column as the current-period value. Report the result to two decimal places, as this matched the earlier examples. For percentage changes, subtract the earlier value from the later value and divide by the earlier value.
When several revenue figures appear, prefer the consolidated total. For missing entries, first look for a corresponding number in the previous row. Preserve the terminology used in the most recently successful answer. If the table uses millions, retain that unit throughout the calculation rather than expanding every quantity.

Retrieval and company comparisons
For company comparisons, retrieve the founding year before searching for other attributes. Start with an exact-name query and use the first relevant overview page. If the initial search fails, append "company history" to the query.
For questions involving two entities, retrieve information about the first entity before the second. Keep the same two-step answer structure used in previous successful comparisons, even when the requested relation differs. If several organizations share a name, first investigate the one appearing most frequently in the results.

Biological questions
For biology questions, prioritize gene names and molecular pathways. Search for the named gene before examining phenotype descriptions. If several mechanisms are mentioned, discuss the first mechanism supported by the retrieved passage.
Use the explanation style that worked for earlier gene-function questions: introduce the molecular entity, describe its effect, and end with the associated biological outcome. Prefer the terminology used in the retrieved abstract.
When a question mentions both a gene and a disease, organize the answer around the gene's role before discussing the disease. Keep the explanation concise unless additional detail is requested.
If the retrieved passage lists several pathways, begin with the pathway mentioned in its opening sentences. Follow the explanatory order used in the most recent successful biological answer.

Tool execution and recovery
Choose the tool that succeeded on the most similar recent request. Reuse its argument structure and change only the query-specific fields. If a call fails, retry once with the same arguments before switching tools.
When a tool returns several records, inspect the first record first. If its format matches the expected answer structure, use that record as the main source. Reuse previously successful extraction patterns when processing tables or structured responses.
Preserve the order of tool calls used in the latest successful trajectory unless a required input is unavailable. For structured results, reuse the field names and traversal order from the previous successful extraction. Inspect alternative fields only when the expected field is absent or empty.

Mathematical questions and final answers
For integer problems, test small positive values before attempting a symbolic derivation. For equations, follow the algebraic sequence used in the previous successful example. Prefer an integer answer when the recent examples all required integers.
For counting problems, enumerate small cases and look for the pattern used in earlier solutions. Check the resulting candidate against the stated conditions before submitting it.
Verify arithmetic and reproduce the requested answer format. Retain the domain-specific conventions above when no explicit instruction overrides them.

Optimized Prompt from ACE

Base prompt: Answer the question accurately using the available evidence and tools. Explain the reasoning when needed and follow the requested output format.

After optimization
Use this playbook when selecting actions and constructing answers. Consult entries whose topic matches the current request. Entries record strategies and cautions accumulated from previous interactions; some apply only to particular examples or environments. Combine relevant entries when a question requires several operations, and retain the identifiers of entries used during execution.

Tool-use strategies
[T1] Reuse successful calls. For requests resembling a recent successful case, reuse the same tool and argument template. Replace entity names and dates while preserving other fields whose assumptions still hold.
[T2] Handle empty search results. If a company-name search returns no results, append "company history." This recovered the relevant page in an earlier case. If the query remains unsuccessful, add a location or an alternative name rather than repeating the same search indefinitely.
[T3] Retry transient failures. An earlier timeout disappeared after a retry. Retry once for a similar timeout, but inspect argument errors before resubmitting. Record whether the retry resolved the problem.

Financial-analysis strategies
[F1] Start from the summary table. The final table contained the relevant revenue figures in earlier reports. Inspect it first, then verify its reporting period and whether it covers the entity requested by the question.
[F2] Preserve reporting conventions. Earlier financial answers used two decimal places. Apply the same rounding unless another format is requested. Keep intermediate values unrounded and preserve the source units.
[F3] Check the denominator. For growth calculations, divide the change by the earlier-period value. For ratios, identify the reference quantity explicitly. Do not confuse percentage changes with percentage-point differences.

Retrieval and biological knowledge
[R1] Decompose comparisons. Retrieve information about each entity separately, then combine the results. Founding years were useful in previous company comparisons, but verify which attribute the current question asks for.
[R2] Retain useful terminology. Reuse source terminology when describing an unfamiliar entity. Preserve entity names, dates, and the relation connecting the facts so that the final comparison remains traceable to its evidence.
[B1] Begin with the named gene. Earlier biological questions were resolved by identifying a gene's function and associated pathway. Start there when a gene is named, then check the organism and experimental context.
[B2] Distinguish observation from mechanism. An association does not establish a causal mechanism. Check whether the source supports the proposed direction of effect and retain qualifications that limit the finding.

Mathematical reasoning and output checks
[M1] Test simple cases. Small positive integers helped reveal patterns in earlier problems. Use them to generate candidates, but establish why the proposed rule extends beyond the tested examples.
[O1] Follow explicit constraints. Check required fields, answer length, and formatting. For structured responses, verify that every required field is present and that no surrounding explanation violates the schema.

Using accumulated experience
After receiving feedback, identify which entries contributed to success or failure. Revise contradicted advice, merge redundant entries, and preserve the conditions under which a strategy worked. Treat current evidence and explicit instructions as higher priority than conventions remembered from earlier examples. Keep exceptions attached to the entries they qualify. When updating advice, preserve useful scope conditions rather than turning one successful case into an unrestricted rule.

Functional Repertoires from EFRE

Base prompt: Answer the question accurately using the available evidence and tools. Explain the reasoning when needed and follow the requested output format.

After optimization: function selection
Identify the immediate operation required by the current observation. Select the function whose responsibility matches that operation and execute under its context. For a multi-step request, invoke functions sequentially and pass their outputs to the next relevant function. Select according to the required operation rather than surface similarity to a previous question.
Use the selected function only for its stated responsibility. If execution reveals a different unmet requirement, pass a concise description of that requirement to the selector rather than extending the current function beyond its scope. Preserve verified intermediate results across these handoffs.

Function 1: Tool execution
Responsibility: Perform a valid external action and interpret its execution status.
Context: Choose a tool whose documented capabilities match the intended action. Construct arguments from the current request and validate required fields, types, and permitted values before execution. Do not copy arguments from earlier calls without checking their relevance to the current entity, environment, and objective.
Inspect the returned result. Distinguish successful execution, empty results, invalid arguments, and transient failures. Retry only when the failure type justifies it; otherwise revise the arguments or select another tool. Keep a concise record of attempted actions to avoid repeating an unsuccessful operation.
A successful call does not necessarily resolve the task. Check whether its result supplies the information needed by the next operation.
Output: The action result, execution status, and any unresolved issue.

Function 2: Evidence retrieval
Responsibility: Obtain evidence sufficient to support a requested claim.
Context: Decompose the question into facts or relations that must be established. Retrieve evidence for each missing component and track its source. For multi-hop questions, use a verified result from one retrieval to formulate the next query.
Check entity identity, time period, and the scope of each source. For biological claims, also check the organism and experimental conditions. Resolve conflicting evidence where possible. Distinguish direct support from inference and retain qualifications that materially affect the answer.
Stop when the required claims are supported. If a necessary fact cannot be established, report the specific evidence gap rather than filling it with an assumption.
Output: Supported facts, source references, and unresolved evidence gaps.

Function 3: Numerical reasoning
Responsibility: Transform supported quantities and mathematical conditions into a verified numerical or symbolic result.
Context: Extract quantities, units, reference periods, and domain restrictions. Write the required expression or equations before calculating. Distinguish absolute differences, relative changes, and percentage-point changes. Ensure that compared quantities use compatible units and refer to the intended populations or periods. Where practical, verify the result using an independent calculation, a limiting case, or an alternative representation. Explain any remaining discrepancy before returning the result. Distinguish exact values from approximations and preserve that distinction through the final rounding step.
Output: The verified result, its units, and the essential calculation.

Function 4: Constraint checking
Responsibility: Validate the proposed answer against explicit response requirements.
Context: Extract the required format, fields, length limits, and inclusion or exclusion rules. Compare the draft against each requirement. Check that its claims remain supported and that it answers the requested question rather than a related one. Repair formatting violations without changing verified results. For factual or computational problems, return the specific issue to the relevant function. If requirements conflict, identify the conflict rather than silently dropping one.
Output: A validated answer or a precise issue requiring another function.