GRID: SCALING TASK-AGNOSTIC INFERENCE IN CONTINUAL PROMPT TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Prompt-based continual learning (CL) offers a parameter-efficient way to adapt large language models (LLMs) across task sequences. However, existing methods often rely on task-aware inference and maintain an expanding set of task-specific prompts, leading to (1) severe performance degradation on earlier tasks when task identifiers are unavailable for prompt selection at inference time, and (2) limited scalability as task sequence grows. We propose GRID, a unified framework designed to address these challenges. GRID incorporates an output-space–aware decoding mechanism that enhances backward transfer by leveraging representative inputs and automatic label semantic normalization, alongside a gradient-guided prompt selection strategy that compresses less informative prompts into a single aggregated representation for scalable, memory-efficient continual learning. Extensive experiments on long-sequence and negative-transfer benchmarks show that GRID improves backward transfer, achieves competitive forward transfer, and substantially reduces prompt memory across encoder-decoder and decoder-only architectures, including T5, Qwen, and LLaMA.

1 INTRODUCTION

Continual learning (CL) Van de Ven & Tolias (2019) enables models to learn from a sequence of tasks without retraining from scratch. CL systems build on prior knowledge while adapting to new tasks, which is crucial in dynamic real-world settings. Recent advancements, especially in NLP Wang et al. (2024); Satapara & Srijith (2024), have focused on three paradigms: *regularization-based methods* Li & Hoiem (2017); Kirkpatrick et al. (2017), *rehearsal-based methods* Rebuffi et al. (2017); Sun et al. (2019), and *architecture-based methods* Veniat et al. (2020). While rehearsal-based methods are effective, they are impractical in privacy-sensitive scenarios Kirkpatrick et al. (2017). With the increasing complexity of pre-trained models Wang et al. (2023a), full model fine-tuning has become infeasible. This has led to the rise of parameter-efficient finetuning (PEFT) techniques Ding et al. (2023), such as *prompt tuning* (PT) Lester et al. (2021), which adapts large models by training only a small set of soft prompts, requiring less than 0.01% of the model's parameters.

Building on this, continual prompt tuning (CPT) extends PT to the CL setting, enabling sequential task adaptation without modifying the base model Wang et al. (2022b). Early work maintained a single shared prompt pool queried by input features Wang et al. (2022b), but retention suffered as a single pool struggled to encode knowledge across diverse tasks. Subsequent methods improved retention by introducing task-specific prompts, either through complementary shared and expert prompt Wang et al. (2022a), or by maintaining a dedicated prompt per task, notably ProgPrompt Razdaibiedina et al. (2023), SHLPT Wu et al. (2024), Q-Tuning Guo et al. (2024), achieving strong knowledge preservation.

Although prompt-based CL has received significant attention, two main challenges remain: First, major methods such as ProgPrompt Razdaibiedina et al. (2023), Q-Tuning Guo et al. (2024), and SHLPT Wu et al. (2024) depend on *task-aware inference*: the task identity is provided at test time to select the corresponding prompt. This assumption allows them to report near-zero forgetting, but in practice, task IDs are often unavailable during inference for prompt selection Wang et al. (2024); Liang & Li (2023). In practical NLP systems, models often receive free-form inputs, and deployed continual models must handle mixed inputs from past tasks without explicit task labels, requiring task-agnostic inference. Although vision-based class-incremental learning (CIL) methods

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.

Tran et al. (2025); McDonnell et al. (2023) also consider task-agnostic prediction, they typically assume a shared classification space. However, our setting involves heterogeneous tasks with diverse behaviors and output spaces, making these approaches inapplicable.

Second, existing continual prompt tuning methods struggle to scale efficiently in both time and memory as the prompt queue expands. ProgPrompt Razdaibiedina et al. (2023) and SHLPT Wu et al. (2024) assign a dedicated soft prompt to each task, causing memory to grow linearly with the task sequence. More recent approaches attempt to mitigate this issue by continually updating the same set of prompts over time Wang et al. (2022b) or by using PCA-based eviction Guo et al. (2024). However, these solutions either suffer from redundant prompt accumulation due to the lack of effective pruning or merging mechanisms, or incur significant computational overhead (e.g., repeated SVD during eviction).

Method	Task-Agnostic Inference	Bounded Prompt Memory
ProgPrompt	✗	✗
SHLPT	✗	✗
Q-Tuning	✗	Partial
GRID	✓	✓

Table 1: Comparison of prompt-based CL methods under two practical deployment constraints. GRID jointly supports task-agnostic inference and bounded prompt memory.

As shown in Table 1, existing prompt-based CL methods in NLP fail to simultaneously support task-agnostic inference and a bounded prompt pool. This motivates us to study an important yet underexplored setting for task-agnostic prompt usage:

Task identifiers are unavailable at inference time for prompt selection, and the prompt pool has a fixed capacity. The model therefore cannot rely on task-specific prompt routing or indefinitely store all prompts as the task sequence grows. Instead, it performs prediction using the available prompt pool, while pruning or merging prompts to maintain bounded memory.

This paper focuses on this setting and makes the following contributions.

- We observe that existing prompt-based CL approaches often struggle in task-agnostic prompt-usage scenarios. Our experiments reveal that when task identities are unavailable at inference time for prompt selection, performance on earlier tasks degrades substantially after training on new ones, with models frequently producing incorrect or ambiguous outputs. Without explicit task cues, the model may generate labels from unrelated tasks encountered during pretraining or earlier learning, or hallucinate entirely invalid outputs not belonging to any seen task's label set.
- We introduce **GRID** (Figure 1), a bounded-memory framework for task-agnostic prompt-based continual learning that addresses the above limitations through two key components: (i) a decoding mechanism that leverages representative inputs and output-space-aware decoding to control the output space of pre-trained LLMs, thereby ensuring label consistency and improving backward transfer (BWT) without relying on task IDs for prompt selection; and (ii) a gradient-guided prompt selection strategy that dynamically evaluates prompt usefulness and merges less informative prompts, significantly reducing memory usage while maintaining both forward and backward transfer performance.

2 Related Work

2.1 Continual Learning

Continual learning (CL) involves learning from a sequence of tasks without full access to previous task data, aiming to preserve prior knowledge and enable positive transfer. A key challenge is catastrophic forgetting (McCloskey & Cohen, 1989), where updates to model parameters on new data erode earlier knowledge. CL strategies are categorized into three approaches: memory-based methods store and replay past task data to mitigate forgetting (Shin et al., 2017; Bang et al., 2021); regularization-based methods penalize deviations from important parameters to retain knowledge without stored data (Kirkpatrick et al., 2017; Zenke et al., 2017). Du et al. (2024) adopts a gradient-masking strategy by updating only high-activation model parameters, achieving task-agnostic and rehearsal-free CL; and architecture-based methods expand the model by adding new components for each task (Yoon et al., 2018; Tiwari & Ji, 2026). These methods face scalability issues in large pre-trained models, motivating the development of parameter-efficient CL techniques using lightweight components like prompts and adapters (Xu et al., 2023; Rücklé et al., 2021).

2.2 Continual Prompt Tuning

Prompt tuning (Lester et al., 2021; Li & Liang, 2021; Gu et al., 2021; Wang et al., 2023b) adapts large language models (LLMs) by learning a small set of continuous vectors, or soft prompts, prepended to the input tokens. Unlike full finetuning, it updates only the prompts while freezing model parameters, achieving competitive or superior performance with lower computational and memory cost. Continual prompt tuning (CPT) (Zhu et al., 2022; Yin et al., 2022; Ermis et al., 2022; Wang et al., 2022a) extends this idea to the continual learning (CL) setting, where models adapt to evolving task sequences. A substantial body of work has focused on improving CPT's ability to retain and transfer knowledge, using techniques such as prompt concatenation (Razdaibiedina et al., 2023), parameter sharing (Wang et al., 2022b), and weighted prompt summation (Jiang et al., 2023). However, existing CPT methods often depend on memory buffers (Zhu et al., 2022; Ermis et al., 2022), assume task-aware prompt selection that can break down under task-agnostic inference (Guo et al., 2024), or maintain expanding prompt pools that become less scalable as the task sequence grows (Razdaibiedina et al., 2023; Wang et al., 2022a; Smith et al., 2023).

2.3 Gradient-Based Data Selection

Gradient-based strategies have been extensively studied in data selection and coreset construction (Aljundi et al., 2019b), where the goal is to identify a representative or influential subset of training data (Borovicka et al., 2012; Rolf et al., 2021). Methods such as Coreset Selection (Killamsetty et al., 2021; Mirzasoleiman et al., 2020; Hao et al., 2023) and Gradient Matching (Aljundi et al., 2019b) leverage the similarity or norm of training gradients to preserve performance while reducing dataset sizes. In the context of prompt learning, some recent works have begun to reduce the prompt memory. For example, Q-Tuning (Guo et al., 2024) uses a PCA-based eviction and L2P (Wang et al., 2022b) keeps updating the same set of prompts over time.

3 Background and Challenges

3.1 Problem Setup

We consider a continual learning (CL) setting in which a model encounters a sequence of N tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$, where each task T_i is associated with a labeled dataset $D_i = \{(x_j, y_j)\}_{j=1}^{|D_i|}$. Here, x_j denotes an input instance and $y_j \in \mathcal{Y}_i$ is the corresponding label. The model is built upon a pretrained encoder-decoder or decoder-only language model $f(\cdot;\theta)$, whose parameters θ are kept fixed throughout learning. Rather than finetuning θ, we adapt the model to each task T_i by learning a soft prompt $\mathbf{p}_i \in \mathbb{R}^{l \times d}$, where l denotes the prompt length and d the embedding dimension. After observing tasks $\{T_1, \ldots, T_{t-1}\}$, we maintain a pool of learned prompts $\mathcal{P} \subseteq \{\mathbf{p}_1, \ldots, \mathbf{p}_{t-1}\}$, which serves as a memory of past task adaptations.

When a new task T_t arrives, we initialize a new prompt $\mathbf{p}_t$ and train it using data from D_t, concatenated with the existing prompt queue $\mathcal{P}$. The backbone model $f(\cdot;\theta)$ remains frozen during training; only the new prompt $\mathbf{p}_t$ is updated. Let $\mathcal{P}^{(t)} = \mathcal{P} \cup \{\mathbf{p}_t\}$ denote the prompt configuration used during training for task T_t. The model prediction is then given by $\hat{y} = f(x; \mathcal{P}^{(t)})$. The training objective is to minimize: $\mathcal{L}_t = \mathbb{E}_{(x,y)\sim D_t}[\ell(f(x; \mathcal{P}^{(t)}), y)]$, where $\ell(\cdot)$ is the token-level cross-entropy loss.

3.2 Task-Agnostic Inference

As motivated before, we focus on the following more realistic task-agnostic inference setting:

Definition 1 (Task-Agnostic Inference)**.** *Task identifiers are unavailable at inference time for prompt selection. As a result, the model cannot rely on task-specific prompt selection. Instead, it performs prediction using the entire available prompt pool $\mathcal{P} \subseteq \{\mathbf{p}_1, \ldots, \mathbf{p}_{t-1}\}$, which may optionally include an aggregated prompt derived from previous filtering stages. The concatenated prompt sequence is prepended to the input and passed to the model. The prediction is defined as: $\hat{y} = f(x; \mathcal{P})$.*

The task-agnostic evaluation objective becomes: $\mathcal{L}_{\text{TA}} = \sum_{T_k \in \mathcal{T}_{\text{past}}} \mathbb{E}_{(x,y)\sim D_k} \left[\ell(f(x; \mathcal{P}), y)\right]$. This formulation evaluates how well the concatenated prompt pool enables generalization to earlier tasks

without needing to retrieve or know their corresponding individual prompts. This setting appears in two important CL scenarios: 1) *Online learning*, where a new task arrives and the model must evaluate or detect alignment with previously seen tasks Aljundi et al. (2019a). 2) *Retroactive evaluation*, where a model trained on a task sequence is later evaluated on older tasks without access to their individual prompts Chaudhry et al. (2018); de Masson D'Autume et al. (2019).

Challenges. First, in task-agnostic settings, unconstrained generative decoding often causes *label drift* and *hallucination*, where models generate semantically related but incorrect or syntactically invalid labels or produce unseen labels as likelihood mass spreads over an expanding vocabulary (Table 2). Second, prior methods such as ProgPrompt Razdaibiedina et al. (2023) and SHLPT Wu et al. (2024) rely on human-specified label mappings (e.g., $0 \rightarrow$ "negative"), which vary across datasets and introduce semantic inconsistencies that hinder cross-task generalization (Appendix, Table 24). Third, existing approaches lack explicit task awareness, treating even semantically related tasks independently, leading to overlapping prompt representations and reduced transfer. Finally, under task-agnostic evaluation, prompt memory grows unbounded as predictions rely on the entire prompt pool, with no principled mechanism for pruning or merging redundant prompts Razdaibiedina et al. (2023). Together, these issues motivate the need for a structured and scalable framework for prompt-based continual learning under task-agnostic conditions.

Task	Input	Gold	ProgPrompt
BoolQ	Did you ever lecture at Harvard?	True	`true</s>`
	The imperialist nation wanted to strangle the economy of its colony.	False	`<pad> Fal`
	The neoclassical canon was rooted in traditional European aesthetics.	True	`<pad> Fal`
MNLI	Some of the buildings around the city square … colonial period.	Entail.	`<pad> True`
	The U.S. Army acceded … to keep U.S. forces in place.	Entail.	`<pad> 4.0`

Table 2: Qualitative ProgPrompt failures under task-agnostic inference. `<pad>` denotes generated special-token noise; blue indicates label drift, and red indicates outputs outside the valid label space.

Why Not Vision Class-Incremental Learning (CIL)?

In this section, we clarify why existing vision class-incremental learning (CIL) methods Smith et al. (2023); McDonnell et al. (2023); Wu et al. (2002) do not directly address our setting. Vision CIL is built on a fundamentally different problem formulation: each task introduces new object classes into a shared classification space, and the model incrementally expands its ability to distinguish among all previously seen classes Bako & Kalita (2026). As a result, the primary challenge is preserving discriminative decision boundaries over an enlarging label set under a unified classifier head.

In contrast, the NLP continual learning setting often involves heterogeneous tasks with substantially different objectives, input structures, label semantics, and output spaces, such as sentiment analysis, natural language inference, and question answering Bako & Kalita (2026); Wu et al. (2024). Unlike vision CIL, these tasks cannot be naturally unified as class expansion within a single shared label space. Consequently, task-agnostic inference in our setting is not merely a class-selection problem. Instead, the model must implicitly infer the underlying task type and activate the appropriate task behavior and output space without access to a task identifier. This structural mismatch makes the direct application of vision CIL methods to NLP task-agnostic continual learning non-trivial.

4 METHOD

To address the aforementioned challenges, we propose a unified framework called **GRID**: **G**radient-based prompt selection with **R**epresentative sample selection, label **I**dentification, and constrained **D**ecoding. GRID integrates two complementary components: 1) An input pipeline that enhances backward retention and output consistency by selecting representative samples, performing task identification, and applying constrained decoding; 2) A gradient-based prompt scoring mechanism that reduces prompt pool size by identifying and merging less informative prompts while preserving relevant task knowledge. Figure 1 illustrates the overall GRID pipeline, while Algorithm 1 details the Prompt Selection and Aggregation procedure. Additional details are provided in Appendix A.

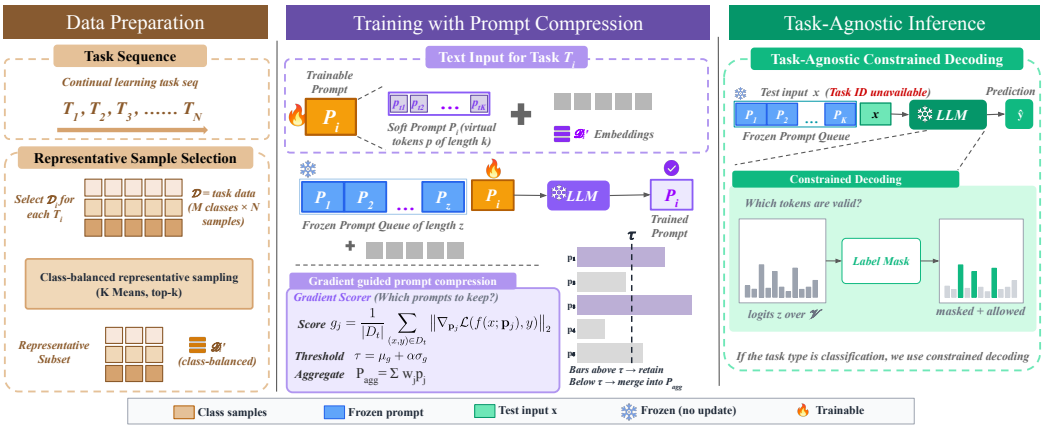


Figure 1: Overview of GRID. Given a continual task sequence, GRID first selects representative samples for each task, then trains a task-specific soft prompt while keeping previously learned prompts frozen. After each task, gradient-guided prompt compression scores the frozen prompt queue and retains or merges prompts based on their contribution to the current task. At inference time, GRID uses the compressed prompt queue without requiring a task identifier, and optionally applies constrained decoding for finite-output tasks.

4.1 Input Construction for Stable Decoding

In task-agnostic inference, unconstrained decoding often leads to *label drift*, where predictions correspond to unrelated tasks or pretraining artifacts. To address this, we reformulate task-agnostic inference over a refined input space built from three components:

1) Representative Input Sampling. To construct a compact and informative training subset, we select k representative samples per class via clustering, improving upon the random sampling strategy used in prior work (Razdaibiedina et al., 2023). The data set $\mathcal{D} = \{(x_j, y_j)\}$ is partitioned by label $y \in \mathcal{Y}$, and the sentences $\mathbf{e}_j = f_{\text{embed}}(x_j)$ are calculated using a pre-trained embedding model (`all-MiniLM-L6-v2`; (Reimers & Gurevych, 2019)). K-Means clustering is performed within each class to ensure intra-class diversity, and the top k/C samples closest to each cluster center are selected based on cosine similarity: $\text{sim}(\mathbf{e}, \mathbf{c}) = \frac{\mathbf{e}\cdot\mathbf{c}}{\|\mathbf{e}\|\|\mathbf{c}\|}$. This yields a balanced subset $\mathcal{D}_{\text{rep}}$ that spans the semantic space of each class.

Algorithm 1: Prompt Selection and Aggregation

Input: Prompt pool $\mathcal{P} = \{\mathbf{p}_j\}$, task data D_t, thresholds α, β
Output: Updated prompt pool $\mathcal{P}'$

for each $\mathbf{p}_j \in \mathcal{P}$ **do**
  Compute gradient score g_j — Eq. 1
end for
Compute μ_g, σ_g and threshold τ — Eq. 2
Split $\mathcal{P}$ into $\mathcal{P}_{\text{high}}$ and $\mathcal{P}_{\text{low}}$ — Eq. 3
if $\mathcal{P}_{\text{low}} \neq \emptyset$ **then**
  Compute aggregate prompt $\mathbf{p}_{\text{agg}}$ — Eq. 4
  $\mathcal{P}' \leftarrow \mathcal{P}_{\text{high}} \cup \{\mathbf{p}_{\text{agg}}\}$
else
  $\mathcal{P}' \leftarrow \mathcal{P}_{\text{high}}$
end if
$\mathcal{P}' \leftarrow \mathcal{P}' \cup \{\mathbf{p}_t\}$
return $\mathcal{P}'$

2) Label Identification. Many datasets lack explicit task metadata and use non-descriptive labels (e.g., {0,1}, {choice1, choice2}), which are ambiguous in task-agnostic inference. We address this with a hierarchical label identification module that infers the task type t^* from an input x and candidate label set $\mathcal{Y}_i$. First, rule-based heuristics match label tokens and input structure to known task templates (e.g., sentiment or NLI). If unsuccessful, we apply zero-shot task classification using a lightweight generative model (e.g., Phi-3.5) to infer the task and semantically remap labels: $(t^*, \tilde{\mathcal{Y}}_i) = \textbf{LLM}(x, \mathcal{Y}_i)$. This normalization enables consistent label semantics and supports constrained decoding in the task-agnostic setting.

3) Constrained Decoding. With remapped label set $\mathcal{L}_i = \{\ell_1, \ldots, \ell_K\}$, we apply constrained decoding at inference. At each decoding step t, we restrict the softmax to only allow tokens from $\mathcal{L}_i$:

$$\tilde{P}(y_t \mid y_{<t}, x) = \text{softmax}(M \odot \mathbf{z}_t)$$
$$M_j = \mathbf{1}[v_j \in \mathcal{L}_i],$$

where $\mathbf{z}_t$ are the raw logits and M is a binary mask over the vocabulary $\mathcal{V}$.

4.2 Prompt Pool Compression via Gradient-Guided Selection

Prompt-based CL methods typically allocate one prompt per task, causing the prompt pool to grow as $\mathcal{O}(N)$. To achieve scalable long-horizon continual learning, we propose a dynamic selection–compression strategy based on gradient relevance. Let $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{t-1}\}$ be the prompt pool before task T_t. For each $\mathbf{p}_j \in \mathcal{P}$, we compute the average gradient norm over new task data D_t:

$$g_j = \frac{1}{|D_t|} \sum_{(x,y)\in D_t} \|\nabla_{\mathbf{p}_j} L_t(f(x; \mathbf{p}_j), y)\|_2. \tag{1}$$

A large g_j indicates that the task substantially updates $\mathbf{p}_j$, suggesting distinct knowledge worth preserving. Conversely, small g_j values imply redundancy with the current task. Prompts are partitioned using the threshold

$$\tau = \mu_g + \alpha\sigma_g, \tag{2}$$

where μ_g, σ_g are the mean and standard deviation of $\{g_j\}$, and α is a tunable hyperparameter; for sensitivity analysis see Appendix §C.6.

$$\begin{aligned} \mathcal{P}_{high} &= \{\mathbf{p}_j : g_j > \tau\}, \\ \mathcal{P}_{low} &= \{\mathbf{p}_j : g_j < \tau\}. \end{aligned} \tag{3}$$

Although equation 1 only measures prompt-task interaction, we observed that low-gradient prompts are often highly redundant (average cosine similarity ≥ 0.87, Euclidean radius $R < 0.45$). When similarity is high, discarding them has little effect; but when $\mathcal{P}_{low}$ is diverse, removal risks losing transferable information. To mitigate this, we aggregate $\mathcal{P}_{low}$ into a single embedding $\mathbf{p}_{agg}$ using gradient-weighted averaging:

$$\mathbf{p}_{agg} = \sum_{\mathbf{p}_j \in \mathcal{P}_{low}} w_j \mathbf{p}_j, \; w_j = \frac{g_j}{\sum_{\mathbf{p}_k \in \mathcal{P}_{low}} g_k}. \tag{4}$$

This ensures that relatively more informative low-gradient prompts contribute proportionally. The updated pool for task T_t becomes $\mathcal{P}' = \mathcal{P}_{high} \cup \{\mathbf{p}_{agg}\}$. This mechanism re-evaluates the existing prompts, retains high-gradient prompts, and merges low-gradient redundant ones into a single aggregated prompt which preserves critical knowledge while substantially reducing memory and inference costs. *Notably, experiments show minimal degradation under compression, confirming that low-gradient prompts contribute little to future tasks.* Figure 2 illustrates this compression process for a task sequence.

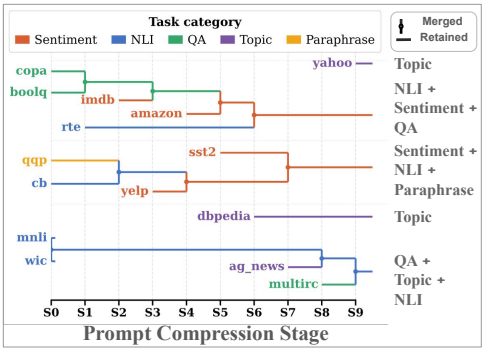


Figure 2: Prompt pool compression trajectory under GRID for Order L1. Each leaf denotes a task-specific prompt, with colors indicating task categories. Branches show prompt consolidation during compression, while isolated branches represent prompts retained independently. GRID groups related prompts while preserving distinct ones when merging is less appropriate.

5 Experiments

We evaluate on six long-sequence benchmarks (L1–L6, 15 tasks each), where L1–L3 follow prior work Razdaibiedina et al. (2023), L4–L6 are newly constructed to study task difficulty progressions (easy→hard, hard→easy, and mixed), and NT1–NT3 form a Negative Transfer benchmark of nine deliberately dissimilar tasks. To evaluate GRID beyond standard classification-style benchmarks, we introduce **MTCL-Bench**, a 15-task Math-Tool continual learning benchmark combining Competition MATH, GSM8K Hendrycks et al. (2021); Cobbe et al. (2021), and seven category-level ToolAlpaca tasks Tang et al. (2023). Construction details are in Appendix §B.1.

6 Results and Discussion

6.1 Average Accuracy across Tasks

GRID achieves strong performance in long-sequence settings (L1–L6; Table 13, 4), reaching 79.2% average accuracy on T5-large and outperforming competitive baselines, including SHLPT, the cur-

(a) Long-sequence accuracy

Method	L1	L2	L3	L4	L5	L6	R	Avg.
Finetune	8.3	8.7	7.8	7.9	8.1	8.9	✗	8.3
Prompt Tuning	8.8	9.5	8.1	9.3	9.4	9.4	✗	9.1
Data Replay	56.2	54.3	53.5	54.8	54.2	55.3	✓	54.7
LFPT5	70.8	69.2	69.4	68.2	69.4	68.5	✓	69.3
Per-task Prompt	75.0	75.6	76.2	74.8	75.9	73.6	✗	75.2
ProgPrompt	75.7	78.6	74.3	75.0	77.1	75.4	✗	76.0
SHLPT	77.4	77.9	78.8	78.4	78.2	**76.2**	✗	77.8
GRID	**79.1**	**80.7**	**81.1**	**79.0**	**79.8**	75.5	✗	**79.2**

(b) MTCL-Bench

Model	Method	CL	Acc.↑	BWT↑
LLaMA-3.1 8B	Pretrained	✗	45.90	–
	Per-task SFT	✗	51.53	–
	Prompt Tuning	✓	16.89	-42.67
	ProgPrompt	✓	49.15	-35.09
	GRID	✓	**50.55**	**-25.34**
Qwen3 4B	Pretrained	✗	45.16	–
	Per-task SFT	✗	55.97	–
	Prompt Tuning	✓	14.12	-7.34
	ProgPrompt	✓	**54.85**	-6.12
	GRID	✓	53.63	**-5.14**

Table 3: Main continual-learning results. **(a)** Average test accuracy (%) on long-sequence orders L1–L6 using T5-large; R indicates data replay. **(b)** MTCL-Bench results on LLaMA-3.1-8B and Qwen3-4B. Pretrained and per-task SFT are non-continual references, while Prompt Tuning, ProgPrompt, and GRID are continual methods. Within the continual methods, **bold** and underlined denote the best and second-best results.

Order	Method	T5-Large			T5-3B			Qwen-3-4B			LLaMA-2-7B		
		Acc↑	BWT↑	FTC↓	Acc↑	BWT↑	FTC↓	Acc↑	BWT↑	FTC↓	Acc↑	BWT↑	FTC↓
L1	ProgPrompt	75.70	-0.7275	77	76.10	-0.8475	80	79.70	-0.8987	82	78.98	-0.8575	80
	SHLPT	77.40	-0.6123	64	78.10	-0.6954	70	80.95	-0.7265	75	79.40	-0.6923	73
	GRID	**79.10**	**-0.3243**	**11**	**80.13**	**-0.3832**	**14**	**83.54**	**-0.4243**	**18**	**81.98**	**-0.3844**	**17**
L2	ProgPrompt	78.60	-0.7625	72	79.56	-0.8054	78	**83.56**	-0.8421	81	81.87	-0.8121	78
	SHLPT	77.90	-0.6870	58	79.33	-0.7458	65	82.68	-0.7970	69	82.12	-0.7645	70
	GRID	**80.70**	**-0.3336**	**5**	**81.40**	**-0.3726**	**9**	82.32	**-0.4198**	**15**	**83.12**	**-0.4387**	**11**
L3	ProgPrompt	74.30	-0.6137	80	76.76	-0.6932	85	78.32	-0.6934	89	77.33	-0.6932	87
	SHLPT	78.80	-0.5174	69	79.32	-0.5964	74	80.54	-0.6344	78	80.65	-0.6345	75
	GRID	**81.10**	**-0.3979**	**18**	**81.65**	**-0.4279**	**19**	**83.87**	**-0.4812**	**23**	**84.76**	**-0.4954**	**22**
L4	ProgPrompt	75.05	-0.6257	87	77.43	-0.6657	89	81.54	-0.6957	92	80.21	-0.6832	89
	SHLPT	78.40	-0.5042	73	79.87	-0.5542	78	81.42	-0.6021	85	81.43	-0.5943	81
	GRID	**79.00**	**-0.3912**	**26**	**80.43**	**-0.4321**	**29**	**82.43**	**-0.4842**	**34**	**81.53**	**-0.4565**	**33**
L5	ProgPrompt	77.10	-0.6351	71	78.43	-0.6790	74	80.21	-0.7254	79	80.43	-0.7187	79
	SHLPT	78.20	-0.4187	58	79.43	-0.4954	63	82.76	-0.5644	68	81.64	-0.5698	75
	GRID	**79.80**	**-0.2956**	**10**	**79.43**	**-0.3576**	**15**	**83.43**	**-0.3588**	**19**	**82.54**	**-0.3654**	**20**
L6	ProgPrompt	75.46	-0.6416	85	77.54	-0.6754	89	81.64	-0.6926	92	79.65	-0.6837	88
	SHLPT	**76.20**	-0.5840	62	**78.43**	-0.6425	67	**82.54**	-0.7736	67	**81.65**	-0.7653	66
	GRID	75.50	**-0.3512**	**13**	77.35	**-0.3753**	**16**	81.26	**-0.4132**	**18**	80.65	**-0.4036**	**18**
Avg	ProgPrompt	76.00	-0.6677	78.7	77.64	-0.7277	82.5	80.83	-0.7580	85.8	79.75	-0.7414	83.5
	SHLPT	77.80	-0.5539	64.0	79.08	-0.6216	69.5	81.82	-0.6830	73.7	81.15	-0.6701	73.3
	GRID	**79.20**	**-0.3490**	**13.8**	**80.06**	**-0.3915**	**17.0**	**82.81**	**-0.4303**	**21.2**	**82.43**	**-0.4240**	**20.2**

Table 4: Performance under task-agnostic prompt usage across task orders (L1–L6). We report only the strongest ProgPrompt and SHLPT baselines and our GRID method for brevity. We report Avg. Accuracy (Acc↑; higher is better), Backward Transfer (BWT↑; less negative is better), and Forgotten Task Count (FTC↓; lower is better) for each backbone. Best and second-best are **bold** and underlined.

rent state of the art. These gains indicate that gradient-guided prompt compression reduces redundancy while preserving task-relevant knowledge. Notably, the improvements extend to decoder-only models (Table 4), with consistent gains on `Qwen-3-4B` and `LLaMA-2-7B`, demonstrating robustness across architectures and decoding paradigms. On the Negative Transfer benchmarks (NT1–NT3; Table 21), GRID continues to outperform baselines (e.g., +3.8% on FT5-base, +3.6% on FT5-large), highlighting its effectiveness under low task similarity. On MTCL-Bench, GRID remains competitive (Table 3), improving average accuracy on LLaMA while maintaining stronger retention than ProgPrompt on both backbones.

6.2 Backward Transfer Analysis

Table 4 reports BWT across long-sequence settings and model variants. GRID consistently exhibits substantially less negative BWT than ProgPrompt and SHLPT, reducing forgetting by nearly half on average, with relative improvements exceeding 50% across all seq-to-seq model variants and similar trends on decoder-only backbones. While larger models achieve higher accuracy (Table 4, 19), they suffer greater forgetting, whereas smaller models retain prior knowledge better, likely due to less aggressive adaptation. MTCL-Bench further confirms GRID's retention benefit: BWT improves from -35.09 to -25.34 on LLaMA and from -6.12 to -5.14 on Qwen (Table 3).

Figure 3 visualizes BWT dynamics for Order L1, where GRID (C) shows brighter lower-triangle regions than ProgPrompt (A) and SHLPT (B), with difference heatmaps (D,E) highlighting widespread gains particularly for early tasks. Figure 16 further shows consistently less negative BWT as tasks accumulate (a), and substantial per-task gains on challenging tasks such as `boolq`, `sst2`, and `dbpedia_14` (b); Figure 15 explains higher forgetting on outlier tasks like `wic` through their distinct semantic cluster. Additional comparisons across L1–L6 and task-order analyses are provided in Appendix §C.1 and Table 14.

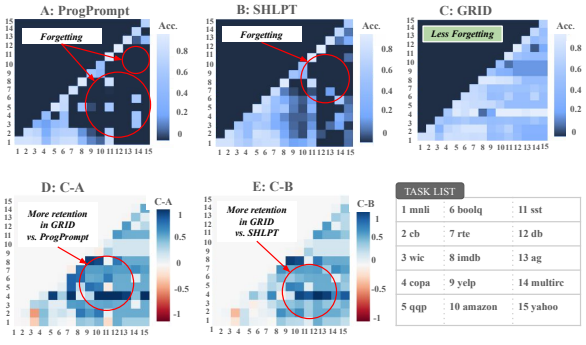


Figure 3: Heatmaps of backward transfer scores on previous tasks for Order L1. (A) Progressive Prompts, (B) SHLPT, (C) GRID, and differences (D) C–A, (E) C–B.

6.3 Forgotten Task Count

Model	Set.	L1	L2	L3	L4	L5	L6	Avg.↓	Imp.↑
T5s	PP	85	85	88	90	84	93	87.5	76.8
	GRID	**14**	**7**	**28**	**36**	**16**	**21**	**20.3**	
FT5s	PP	93	95	95	94	97	93	94.5	76.7
	GRID	**17**	**14**	**28**	**26**	**20**	**27**	**22.0**	
T5b	PP	88	82	76	90	83	88	84.5	80.9
	GRID	**15**	**6**	**19**	**29**	**13**	**15**	**16.2**	
FT5b	PP	94	93	93	93	93	93	93.2	83.0
	GRID	**10**	**10**	**15**	**25**	**14**	**21**	**15.8**	
T5l	PP	77	72	80	87	71	85	78.7	82.4
	GRID	**11**	**5**	**18**	**26**	**10**	**13**	**13.8**	
FT5l	PP	92	93	93	93	91	93	92.5	79.8
	GRID	**15**	**10**	**24**	**24**	**14**	**25**	**18.7**	

Table 5: Forgotten-task counts for ProgPrompt (PP) and GRID across L1–L6. Avg.↓ denotes the average forgotten-task count, while Imp.↑ denotes GRID's percentage reduction over PP.

We define the *Forgotten Task Count (FTC)* as the number of tasks whose accuracy drops below a fixed fraction of their standalone performance. Formally, task T_i is considered forgotten at step t if $a_i^{(t)} < \tau \cdot \min_j a_j^{(0)}$, where $a_i^{(0)}$ denotes standalone accuracy and $\tau \in (0, 1)$. Using zero accuracy as a cutoff is misleading, as performance may degrade to small but nonzero values; our criterion instead enforces a principled absolute threshold[1]. GRID substantially reduces forgetting compared to ProgPrompt and SHLPT, forgetting on average only 13.8 tasks versus 78.7 and 64.0, respectively. These results demonstrate that GRID mitigates catastrophic forgetting in long-horizon continual learning while preserving task-relevant knowledge in a compact form. Results across model variants are reported in Table 5.

6.4 Scalability to Larger Models

Results on larger models (Table 4) show that GRID preserves strong accuracy and reduced forgetting at scale. Figure 4 further demonstrates that this behavior remains stable as model size increases, indicating that GRID scales effectively to large language models.

6.5 Training and Inference Overhead.

GRID introduces a one-time gradient-scoring step per retained prompt when a new task arrives, incurring $O\left(\sum_{t=1}^{N} |P_t|\right)$ total overhead across N tasks. Empirically, this adds ∼12.9–15.6 minutes

[1] Amazon had the lowest standalone accuracy (∼0.50); setting $\tau = 0.4$ yields an absolute threshold of 0.20.

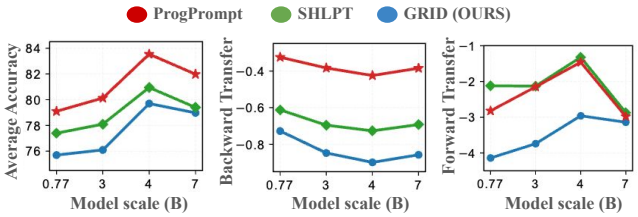


Figure 4: Effect of model scaling across T5-Large (0.77B), T5-XL (3B), Qwen-3 (4B), and LLaMA-2 (7B).

per task (2.5–3.1 min/prompt $\times$ $\sim$5 prompts), totaling $\sim$144 minutes across Tasks 6–15. Overall wall-clock time remains comparable to ProgPrompt (27 vs. 24 h on A100; 12 vs. 11 h on H100). At inference, retaining $\sim$5 prompts instead of 15 yields a 5–20% speedup for long inputs ($\sim$2048 tokens) and 30–50% for shorter inputs ($\sim$256 tokens). Full details appear in Appendix §D.

6.6 Ablation Study

We conduct an ablation on task orders L1–L3, evaluating BWT across four GRID variants that remove gradient-based selection (G), constrained decoding (D), representative input selection (R), or all components (Table 6). The full model achieves the best BWT on both T5-large and Qwen-3-4B, with results indicating constrained decoding as the primary contributor to retention, while gradient-based selection mainly improves scalability (200KB vs. 600KB for ProgPrompt). Detailed ablation in Appendix §C.7

Model	Variant	BWT	Mem.	GPU
T5-L	**GRID**	**−0.3511**	**200**	27:08
	w/o G	−0.3490	600	25:35
	w/o G,D	−0.6863	600	26:42
	w/o G,D,R	−0.6954	600	24:58
	w/o all	−0.7012	600	23:58
Qwen	**GRID**	**−0.4418**	**200**	84:12
	w/o G	−0.4393	600	81:38
	w/o G,D	−0.7958	600	81:35
	w/o G,D,R	−0.8009	600	80:46
	w/o all	−0.8114	600	78:47

Table 6: GRID ablation on L1–L3. BWT is averaged across orders; Mem. is prompt memory (KB), and GPU reports runtime (h:m).

7 Conclusion

In this work, we introduce GRID, a unified framework that makes prompt-based continual learning both scalable and robust to forgetting. Unlike prior approaches that depend on task-aware inference or accumulate ever-growing prompt pools, GRID tackles two key challenges: *forgetting under task-agnostic inference* and the *inefficiency of unbounded prompt memory*. It integrates *constrained decoding* with *gradient-guided prompt selection and compression*, enabling consistent label generation, compact memory, and improved retention, while substantially enhancing backward transfer and preserving competitive forward transfer.

AI Use Statement

In preparing this work, we used generative AI tools to assist with language revision, including improving the grammar, clarity, and presentation of author-written text. All AI-assisted suggestions were independently reviewed and verified by the authors, and no AI-generated content was accepted without author evaluation. The authors take full responsibility for the final content of the paper.

Reproducibility Statement

We provide anonymized source code, training and evaluation configurations, and data-preparation scripts required to reproduce our experiments. Additional implementation details, hyperparameters, datasets, and evaluation protocols are described in the main paper and appendix.

References

Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval, 2019a. URL https://arxiv.org/abs/1908.04742.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019b.

John Bako and Jugal Kalita. Addressing catastrophic forgetting in class-incremental learning—a survey. 2026.

Jihwan Bang, Heesu Kim, YoungJoon Yoo, Jung-Woo Ha, and Jonghyun Choi. Rainbow memory: Continual learning with a memory of diverse samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8218–8227, 2021.

Tomas Borovicka, Marcel Jirina Jr, Pavel Kordik, and Marcel Jirina. Selecting representative data sets. *Advances in data mining knowledge discovery and applications*, 12:43–70, 2012.

Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. *Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence*, pp. 556–572. Springer International Publishing, 2018. ISBN 9783030012526. doi: 10.1007/978-3-030-01252-6_33. URL http://dx.doi.org/10.1007/978-3-030-01252-6_33.

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

Cyprien de Masson D'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

Wenyu Du, Shuang Cheng, Tongxu Luo, Zihan Qiu, Zeyu Huang, Ka Chun Cheung, Reynold Cheng, and Jie Fu. Unlocking continual learning abilities in language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6503–6522, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.379. URL https://aclanthology.org/2024.findings-emnlp.379/.

Beyza Ermis, Giovanni Zappella, Martin Wistuba, Aditya Rawal, and Cedric Archambeau. Memory efficient continual learning with transformers. *Advances in Neural Information Processing Systems*, 35:10629–10642, 2022.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. Ppt: Pre-trained prompt tuning for few-shot learning. *arXiv preprint arXiv:2109.04332*, 2021.

Yanhui Guo, Shaoyuan Xu, Jinmiao Fu, Jia Liu, Chaosheng Dong, and Bryan Wang. Q-tuning: Queue-based prompt tuning for lifelong few-shot language learning. *arXiv preprint arXiv:2404.14607*, 2024.

Jie Hao, Kaiyi Ji, and Mingrui Liu. Bilevel coreset selection in continual learning: A new formulation and algorithm. *Advances in Neural Information Processing Systems*, 36:51026–51049, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Gangwei Jiang, Caigao Jiang, Siqiao Xue, James Y Zhang, Jun Zhou, Defu Lian, and Ying Wei. Towards anytime fine-tuning: Continually pre-trained language models with hypernetwork prompt. *arXiv preprint arXiv:2310.13024*, 2023.

Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training, 2021. URL https://arxiv.org/abs/2103.00123.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Yan-Shuo Liang and Wu-Jun Li. Loss decoupling for task-agnostic continual learning. *Advances in Neural Information Processing Systems*, 36:11151–11167, 2023.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6470–6479, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Mark D McDonnell, Dong Gong, Amin Parvaneh, Ehsan Abbasnejad, and Anton Van den Hengel. Ranpac: Random projections and pre-trained models for continual learning. *Advances in Neural Information Processing Systems*, 36:12022–12053, 2023.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models, 2020. URL https://arxiv.org/abs/1906.01827.

SJ—Yang Pan. Q.: A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

Chengwei Qin and Shafiq Joty. Lfpt5: A unified framework for lifelong few-shot language learning based on prompt tuning of t5. *arXiv preprint arXiv:2110.07298*, 2021.

Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. Progressive prompts: Continual learning for language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UJTgQBc91_.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Esther Rolf, Theodora T Worledge, Benjamin Recht, and Michael Jordan. Representation matters: Assessing the importance of subgroup allocations in training data. In *International Conference on Machine Learning*, pp. 9040–9051. PMLR, 2021.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the efficiency of adapters in transformers. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7930–7946, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.626. URL https://aclanthology.org/2021.emnlp-main.626/.

Shrey Satapara and P. K. Srijith. TL-CL: Task and language incremental continual learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 12123–12142, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.676. URL https://aclanthology.org/2024.emnlp-main.676/.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11909–11919, 2023.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. Lamol: Language modeling for lifelong language learning. *arXiv preprint arXiv:1909.03329*, 2019.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases, 2023.

Anushka Tiwari and Kaiyi Ji. Turning back without forgetting: Selective backward refinement for parameter-efficient continual learning. *arXiv preprint arXiv:2606.01379*, 2026.

Quyen Tran, Tung Lam Tran, Khanh Doan, Toan Tran, Dinh Phung, Khoat Than, and Trung Le. Boosting multiple views for pretrained-based continual learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

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

Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. Pre-trained language models and their applications. *Engineering*, 25:51–65, 2023a.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. Multitask prompt tuning enables parameter-efficient transfer learning. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=Nk2pDtuhTq.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European conference on computer vision*, pp. 631–648. Springer, 2022a.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 139–149, 2022b.

Zirui Wang, Sanket Vaibhav Mehta, Barnabas Poczos, and Jaime Carbonell. Efficient meta lifelong-learning with limited memory. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 535–548, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.39. URL https://aclanthology.org/2020.emnlp-main.39/.

Chenyuan Wu, Gangwei Jiang, and Defu Lian. Mitigate negative transfer with similarity heuristic lifelong prompt tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10944–10959, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.650. URL https://aclanthology.org/2024.findings-acl.650/.

Dekai Wu, Grace Ngai, Marine Carpuat, Jeppe Larsen, and Yongsheng Yang. Boosting for named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*, 2002. URL https://aclanthology.org/W02-2035/.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023. URL https://arxiv.org/abs/2312.12148.

Wenpeng Yin, Jia Li, and Caiming Xiong. ConTinTin: Continual learning from task instructions. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3062–3072, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.218. URL https://aclanthology.org/2022.acl-long.218/.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Sk7KsfW0-.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. PMLR, 2017.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Qi Zhu, Bing Li, Fei Mi, Xiaoyan Zhu, and Minlie Huang. Continual prompt tuning for dialog state tracking. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1124–1137, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.80. URL https://aclanthology.org/2022.acl-long.80/.

CONTENTS OF THE APPENDIX

A Algorithmic Details **15**

B Full Experimental Setup and Configurations **15**

B.1 Task Sequence Orders and Benchmark Construction 15

B.2 Detailed Experimental Configurations 17

B.2.1 Baselines Details. 17

B.2.2 Models. 18

B.2.3 Prompt Initialization Strategies. 18

B.2.4 Representative sample selection. 18

C Additional Results **18**

C.1 Impact of Task Order on Backward Transfer 18

C.2 Forward transfer on long-sequence benchmarks. 19

C.3 Analysis of Results on Negative Transfer Benchmarks. 19

C.4 Extension to Open-Ended Generation (SuperNI Benchmark). 19

C.5 Results on MTCL-Bench 19

C.6 Hyperparameter Sensitivity: Prompt Selection Threshold α. 20

C.7 Detailed Ablation Study 20

C.7.1 Impact of Prompt Selection Strategies (Gradient vs. FIFO vs. Random). 21

C.7.2 Effect of Sample Size and Clustering on Performance 21

C.7.3 Effect of Prompt Length. 21

C.7.4 Effect of Queue Size. 22

D Detailed Training and Inference Overhead Analysis. **22**

D.1 Complexity analysis 22

D.2 Runtime and Hardware. 22

D.3 Training overhead analysis 22

D.4 Inference efficiency. 23

A Algorithmic Details

Algorithm 2 presents the representative sample selection procedure used to construct compact yet diverse task-specific training subsets.

Algorithm 2 Representative Sample Selection

Input: Dataset $\mathcal{D} = \{(x_j, y_j)\}$, samples per class k, embedding model f_{embed}, number of clusters C
Output: Subset $\mathcal{D}_{\text{rep}}$

```
Initialize D_rep ← ∅
for each label y do
    D_y ← {x_j | y_j = y}
    Embed each x_j: e_j ← f_embed(x_j)
    Run K-Means on {e_j} into C clusters
    for each cluster c do
        Select top k/C samples by similarity to center
        Add to D_rep
    end for
    if selected ¡ k then
        Add random samples to reach k total
    end if
end for
Shuffle and return D_rep
```

B Full Experimental Setup and Configurations

Dataset	Category	Task	Domain	Metric
Yelp	CL benchmark	sentiment analysis	Yelp reviews	acc
Amazon	CL benchmark	sentiment analysis	Amazon reviews	acc
DBpedia	CL benchmark	topic classification	Wikipedia	acc
Yahoo	CL benchmark	QA	Yahoo Q&A	acc
AG News	CL benchmark	topic classification	news	acc
MNLI	GLUE	NLI	various	acc
QQP	GLUE	paraphrase detection	Quora	acc/F1
RTE	GLUE	NLI	news, Wikipedia	acc
SST-2	GLUE	sentiment analysis	movie reviews	acc
WiC	SuperGLUE	word sense disambiguation	lexical db	acc
CB	SuperGLUE	NLI	various	acc
COPA	SuperGLUE	QA	blogs, encyclopedia	acc
BoolQ	SuperGLUE	boolean QA	Wikipedia	acc
MultiRC	SuperGLUE	QA	various	F1/EM
IMDB	Other	sentiment analysis	movie reviews	acc

Table 7: Overview of the 15 datasets used in our CL experiments, including their evaluation metrics. Datasets from CL benchmark (Zhang et al., 2015), GLUE (Wang et al., 2018), and SuperGLUE (Wang et al., 2019) were utilized, along with the IMDB movie reviews dataset. For tasks with two evaluation metrics, we report the average as the final performance measure.

B.1 Task Sequence Orders and Benchmark Construction

Long-Sequence CL Setting: Following the approach of (Razdaibiedina et al., 2023), we consider a total of 15 distinct tasks. These consist of five datasets from the CL benchmark (Zhang et al., 2015): AG News (topic classification), Amazon Reviews (sentiment analysis), Yelp Reviews (sentiment analysis), DBpedia (Wikipedia text classification), Yahoo Answers (Q&A classification), four tasks from the GLUE benchmark (MNLI, QQP, RTE, SST2) (Wang et al., 2018), five tasks from the SuperGLUE benchmark (WiC, CB, COPA, MultiRC, BoolQ) (Wang et al., 2019), and the IMDB movie reviews dataset for sentiment analysis (Maas et al., 2011). Figure 15 visualizes task embedding distributions under two sentence encoders, highlighting semantic clustering and separation among tasks in the sequence. We evaluate our methods across six different task sequence orders. Three of these sequences follow standard long-sequence continual learning orders that have been used in prior works (e.g., L1–L3) (Razdaibiedina et al., 2023; Guo et al., 2024). We additionally

#	Dataset name	Task	Metric
1	task639	dialogue generation	Rouge-L
2	task1590	dialogue generation	Rouge-L
3	task1729	dialogue generation	Rouge-L
4	task181	information extraction	Rouge-L
5	task748	information extraction	Rouge-L
6	task1510	information extraction	Rouge-L
7	task002	question answering	Rouge-L
8	task073	question answering	Rouge-L
9	task591	question answering	Rouge-L
10	task511	summarization	Rouge-L
11	task1290	summarization	Rouge-L
12	task1572	summarization	Rouge-L
13	task363	sentiment analysis	accuracy
14	task875	sentiment analysis	accuracy
15	task1687	sentiment analysis	accuracy

Table 8: Details of the SuperNI benchmark used for open-ended instruction-based continual learning. The benchmark includes dialogue generation, information extraction, question answering, summarization, and sentiment analysis tasks.

Order	Task Sequence
L1	mnli → cb → wic → copa → qqp → boolq → rte → imdb → yelp → amazon → sst2 → dbpedia → ag → multirc → yahoo
L2	multirc → boolq → wic → mnli → cb → copa → qqp → rte → imdb → sst2 → dbpedia → ag → yelp → amazon → yahoo
L3	yelp → amazon → mnli → cb → copa → qqp → rte → imdb → sst2 → dbpedia → ag → yahoo → multirc → boolq → wic
L4	sst2 → imdb → yelp → amazon → ag → yahoo → dbpedia → mnli → rte → cb → qqp → copa → boolq → wic → multirc
L5	multirc → wic → boolq → copa → qqp → cb → rte → mnli → dbpedia → yahoo → ag → amazon → yelp → imdb → sst2
L6	sst2 → copa → ag → imdb → mnli → yahoo → rte → yelp → qqp → cb → amazon → dbpedia → boolq → wic → multirc
NT1	multirc → wic → mnli → cb → rte → qqp → yahoo → yelp → amazon
NT2	amazon → yelp → yahoo → qqp → rte → cb → mnli → wic → multirc
NT3	yahoo → mnli → amazon → cb → yelp → rte → qqp → multirc → wic
MB	math-prealgebra → math-algebra → math-geometry → math-counting-probability → math-number-theory → math-intermediate-algebra → math-precalculus → gsm8k → tool-development → tool-geocoding → tool-games-comics → tool-transportation → tool-finance → tool-video → tool-health

Table 9: Task sequence orders used in our CL experiments. Orders L1–L3 correspond to prior long-sequence CL benchmarks, while L4–L6 are our proposed custom sequences designed to study easy-to-hard, hard-to-easy, and mixed task progressions. NT1–NT3 denote negative-transfer orders constructed from deliberately dissimilar tasks. MTCL-Bench (MB) denotes our Math-Tool continual learning benchmark, which extends evaluation to mathematical reasoning and tool-selection tasks.

propose three novel task orders: (1) *Order L4: easy-to-hard*, where tasks are grouped by category and simpler tasks such as binary sentiment classification are introduced first, followed by more complex reasoning tasks like MultiRC and COPA; (2) *Order L5: hard-to-easy*, which reverses this progression; and (3) *Order L6: mixed*, where tasks from different categories and difficulty levels are randomly interleaved. These variations allow us to assess how task presentation order and semantic

Order	SuperNI Benchmark
1	task1572 → task363 → task1290 → task181 → task002 → task1510 → task639 → task1729 → task073 → task1590 → task748 → task511 → task591 → task1687 → task875
2	task748 → task073 → task1590 → task639 → task1572 → task1687 → task591 → task363 → task1510 → task1729 → task181 → task511 → task002 → task1290 → task875

Table 10: Task orders used for the SuperNI benchmark.

similarity affect forgetting, transfer, and model generalization in long-horizon continual learning scenarios.

Negative Transfer Benchmark. To evaluate the robustness of our approach against negative transfer (Pan, 2010), we introduce a *Negative Transfer Benchmark* (NT1, NT2, NT3) inspired by the SHLPT framework (Wu et al., 2024). Specifically, we identify task pairs that exhibit negative transfer based on the analysis provided in the SHLPT. Using these insights, we create three task sequences that are likely to cause negative transfer, meaning earlier tasks make it harder to learn later ones. This helps us test how well our method handles such challenges.

Tool Category	# Examples	# Tools
Development	298	36
Geocoding	287	32
Games & Comics	276	29
Transportation	215	21
Finance	138	14
Video	124	14
Health	122	12

Table 11: Statistics of the selected ToolAlpaca categories used in the Math–Tool continual learning benchmark.

Math-Tool Continual Learning Benchmark (MTCL-Bench) To extend evaluation beyond classification-style continual learning, we construct a 15-task Math-Tool benchmark. For mathematical reasoning, we use competition_math (Hendrycks et al., 2021) and GSM8K, treating the seven Competition MATH subjects as separate tasks and GSM8K as the eighth task. For each Competition MATH subject, we select 1K examples for training and a separate set of 100 examples for testing. For GSM8K, we sample 1K training examples from the official training split and 100 examples from the official test split. For tool use, we process ToolAlpaca (Tang et al., 2023) into seven category-level tool-selection tasks: Development, Geocoding, Games & Comics, Transportation, Finance, Video, and Health. Each ToolAlpaca instance is converted into a category-level tool-selection example and split using a 90/10 stratified split by category and gold tool; category statistics are shown in Table 11.

B.2 Detailed Experimental Configurations

We report task-agnostic continual prompt-learning results on long-sequence benchmarks (L1–L6) and negative-transfer sequences (NT1–NT3). Unless otherwise noted, we use a prompt length of 10 tokens and k=1000 representative samples per class. We train seq-to-seq models for 10 epochs and autoregressive models for 5 epochs, following the configurations summarized in Table 12.

B.2.1 Baselines Details.

We compare against seven continual-learning baselines:[2] **Finetune** (Wang et al., 2020), **Prompt Tuning** (Lester et al., 2021; Qin & Joty, 2021), **Data Replay** (de Masson D'Autume et al., 2019), **LFPT5** (Qin & Joty, 2021), **Per-task Prompt** (Lester et al., 2021; Qin & Joty, 2021), **ProgPrompt** (Razdaibiedina et al., 2023), and **SHLPT** (Wu et al., 2024). We report average test accuracy, backward transfer (BWT), and forward transfer (FWT) (Lopez-Paz & Ranzato, 2017), averaged over three runs.

For the MTCL-Bench, we additionally compare GRID against four baselines to the new setting: (i) pretrained models evaluated directly on each task without training, (ii) per-task SFT, where a separate LoRA adapter is trained independently for each task to estimate task-specific upper-bound performance, (iii) ProgPrompt, where task prompts are accumulated without pruning or merging, and (iv) continual Prompt Tuning, where a single soft prompt is sequentially updated across tasks.

[2]We do not include Q-Tuning (Guo et al., 2024) due to its incompatible setup and unavailable code under company restrictions.

Argument	Autoregressive	Seq-to-seq
Architecture	Autoregressive	Seq-to-seq
Base model	Qwen3-4B-ins; LLaMA-2-7B, LLaMA-3.1-8B-ins	T5 (and Flan-T5)
Number of tasks	15	15
Prefix length	10	10
Max sequence length	256	256
Learning rate	0.03	0.3
Batch size	16	8
Samples per class (k)	1000	1000
Training epochs	5, 3 (MTCL-Bench)	10
Frozen backbone	Yes	Yes
Evaluation setting	Task-agnostic	Task-agnostic
Fixed test data	Yes	Yes
Early stopping	Enabled	Enabled

Table 12: Training configurations used in our experiments for autoregressive and seq-to-seq backbones.

B.2.2 MODELS.

To demonstrate architectural generality, we evaluate both **seq-to-seq** and **autoregressive** backbones. For seq-to-seq, we use T5 and Flan-T5 across `small`, `base`, `large`, and `3B` variants. For autoregressive evaluation, we use `Qwen/Qwen3-4B-Instruct` and `meta-llama/Llama-2-7b-hf` under the same task-agnostic setup. For MTCL-Bench, which requires stronger mathematical reasoning and tool-selection capabilities, we use `Qwen/Qwen3-4B-Instruct-2507` and `meta-llama/Llama-3.1-8B-Instruct`. Training configuration in Table 12.

B.2.3 PROMPT INITIALIZATION STRATEGIES.

For both architectures, we freeze the backbone parameters and optimize only task-specific soft prompts. For instruction-tuned autoregressive models, prompts are initialized using a short natural-language instruction that enumerates the task label names (e.g., "Classify into: ..."), and the corresponding token embeddings are used to seed the 10-token soft prompt. For seq-to-seq models, prompts are initialized directly in the embedding space. For MTCL-Bench, we use task-type-specific initialization: math prompts are initialized with "Solve the math problem step by step and write the final answer in boxed form," while tool-use prompts are initialized with "Select the best tool from the candidate tool list for the user request. We also experimented with random prompt initialization and observed a slight but not statistically significant degradation in performance compared to instruction-based initialization.

B.2.4 REPRESENTATIVE SAMPLE SELECTION.

We construct a refined training subset per class using the clustering-based representative selection procedure in Algorithm 2. We embed inputs using `all-MiniLM-L6-v2` and run K-means to select examples closest to cluster centers, ensuring coverage and diversity while keeping a fixed per-class budget.

C ADDITIONAL RESULTS

C.1 IMPACT OF TASK ORDER ON BACKWARD TRANSFER

Table 14 provides a detailed analysis of backward transfer (BWT) across multiple long-sequence task orderings (L1–L6) and model scales. A consistent pattern emerges across all seq-to-seq model variants: **Order L5 (hard-to-easy)** yields the least negative BWT, indicating significantly reduced forgetting compared to **Order L4 (easy-to-hard)**. For identical task sets, L5 improves average BWT by over 50% relative to ProgPrompt, demonstrating that exposing the model to more challenging tasks earlier leads to stronger and more stable prompt representations.

We further observe that **Order L6 (mixed)** mostly outperforms L4 but remains slightly inferior to L5, suggesting that interleaving tasks of varying difficulty offers a compromise between stability

and adaptability. This trend holds across model sizes, with larger models exhibiting higher absolute forgetting but benefiting more from favorable task curricula. Overall, these results confirm that task order plays a critical role in long-horizon task-agnostic continual learning, and that hard-to-easy curricula are particularly effective for mitigating forgetting under prompt-based adaptation. Figures 7 - 14 provide complementary per-task BWT bar plots and heatmaps for Orders L2-L6.

C.2 Forward transfer on long-sequence benchmarks.

Table 22 reports FWT on the long-sequence task orders L1–L6. Overall, GRID achieves better average FWT than Progressive Prompts across most model variants, improving the mean FWT from -4.45 to -2.16 across the six T5/Flan-T5 backbones. The gains are especially clear for Flan-T5-base, where FWT improves from -4.92 to 1.84, and Flan-T5-large, where it improves from -2.42 to -0.09. These results indicate that GRID's reduction in backward forgetting does not come at the cost of forward transfer. Although T5-base shows lower FWT under GRID, the overall trend suggests that prompt-pool compression preserves useful transferable knowledge while reducing interference from redundant prompts.

C.3 Analysis of Results on Negative Transfer Benchmarks.

Tables 20, 21, and 23 report BWT, accuracy, and FWT on the negative transfer sequences NT1–NT3, where tasks are deliberately dissimilar to test cross-task interference.

BWT (Table 20). GRID consistently and substantially reduces forgetting across all model scales. Improvements over PP range from $\sim 51\%$ (T5-large) to $\sim 72\%$ (Flan-T5-small), confirming that constrained decoding and gradient-guided compression together suppress interference even when tasks share little semantic overlap.

Accuracy (Table 21). Results are competitive but mixed. GRID outperforms PP on Flan-T5-small ($+1.6\%$), Flan-T5-base ($+1.2\%$), and matches closely on most others. PP holds a marginal edge on T5-base (-1.1%) and T5-large (-0.4%). The pattern suggests that GRID's accuracy advantage is less pronounced under high task dissimilarity compared to long-sequence settings, likely because constrained decoding provides smaller gains when task boundaries are already well-separated.

FWT (Table 23). Forward transfer is similarly mixed. GRID improves FWT on Flan-T5-small (-5.34 vs. -7.27) and Flan-T5-base (-5.07 vs. -6.44), but underperforms PP on T5-base and T5-large. This indicates that prompt pool compression occasionally discards prompts with transferable signals in low-similarity settings.

C.4 Extension to Open-Ended Generation (SuperNI Benchmark).

We evaluate GRID on the SuperNI benchmark described in Table 10 and Table 8, which contains 15 instruction-following tasks spanning dialogue generation, information extraction, question answering, summarization, and sentiment analysis. Unlike the fixed-label classification benchmarks used in the main experiments, most SuperNI tasks require open-ended generation and are evaluated with Rouge-L, while sentiment tasks are evaluated using accuracy.

As shown in Table 18, GRID improves both average accuracy and BWT over ProgPrompt and SHLPT across two task orders, suggesting that its gains are not solely due to constrained decoding but also arise from reducing cross-task prompt interference. These results suggest that GRID is not limited to fixed-label classification settings: even when constrained label decoding is not applicable, prompt-pool compression improves retention by reducing cross-task prompt interference.

C.5 Results on MTCL-Bench

Table 3 shows that MTCL-Bench is a challenging continual learning benchmark across mathematical reasoning and tool-selection tasks. Although ProgPrompt achieves slightly higher average accuracy on Qwen, GRID consistently provides better task retention across both backbones, as shown by less negative BWT and less forgetting. On LLaMA, GRID improves average accuracy over ProgPrompt while reducing forgetting by a large margin, with BWT improving from -35.09 to -25.34. On Qwen, GRID also improves retention, reducing BWT from -6.12 to -5.14. These results suggest

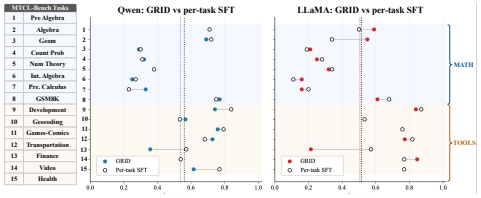


Figure 5: Task-level comparison between GRID and per-task LoRA SFT on MTCL-Bench. Each point reports task-level accuracy for Qwen and LLaMA, with shaded regions separating math and tool-use tasks. Per-task LoRA SFT trains an independent LoRA adapter for each task and serves as a task-specific reference, while GRID uses bounded-memory continual soft-prompt learning.

α	Retained Prompts	BWT ↑	Prompt Mem. (KB) ↓
0.0	7	-0.40	280
0.2	6	-0.35	240
0.4	6	-0.35	240
0.6	5	**-0.34**	**200**
0.8	5	**-0.34**	**200**
1.0	4	-0.38	160
1.2	4	-0.38	160
1.4	3	-0.45	120
1.6	3	-0.45	120

Table 13: Sensitivity to the prompt-selection threshold α. GRID remains stable over a broad range of α, with $\alpha = 0.6$–0.8 providing the best memory–BWT trade-off.

that GRID's selective prompt compression better preserves previously learned task knowledge than naive prompt accumulation, leading to stronger backward transfer and more stable long-horizon retention.

Comparison with per-task LoRA SFT. Figure 5 compares GRID with a per-task LoRA SFT reference on MTCL-Bench. Unlike per-task LoRA SFT, which trains an independent LoRA adapter for each task, GRID learns continually using a bounded soft-prompt pool. GRID remains competitive on several tasks, while the remaining gaps highlight cases where continual prompt learning still suffers from task interference.

C.6 Hyperparameter Sensitivity: Prompt Selection Threshold α.

The hyperparameter α controls the aggressiveness of gradient-based prompt selection, and therefore determines both the number of retained prompts and the prompt memory footprint. As shown in Table 13, GRID exhibits a stable memory–performance trade-off. Smaller values of α retain more prompts, leading to slightly better retention but higher memory usage, whereas larger values increase compression with gradual degradation. Importantly, a broad range of $\alpha \in [0.6, 0.8]$ consistently retains about five prompts ($\sim$200KB) and achieves near-optimal BWT, indicating that GRID is robust and does not require precise tuning. Notably, this retained capacity also matches the best queue size observed in our queue-size analysis, suggesting that gradient-based selection can automatically identify an effective memory budget.

C.7 Detailed Ablation Study

To evaluate the contribution of each component in the GRID framework, we conduct an ablation study on task order L1, L2, and L3, measuring BWT under four variants: (0) the full **G.R.I.D.** model, (1) without gradient-based prompt selection (G), (2) without constrained decoding (D), (3) without representative input selection (R), and (4) without all components. As shown in Table 26, the full model yields the highest BWT for both T5-large (**-0.3511**) and Qwen-3-4B (**-0.4418**). Excluding gradient-based selection has minimal effect on BWT, consistent with its role in reducing memory rather than directly improving retention: GRID requires only **200 KB** of storage versus **600 KB** for ProgPrompt, a **66.7% reduction**. In contrast, removing constrained decoding causes a pronounced BWT drop, and eliminating both components leads to the worst performance (**-0.7012** for T5-large, **-**

Model	Setting	L1	L2	L3	L4	L5	L6	Avg	Imp. (%)
T5-small	PP	-0.6081	-0.5414	-0.5126	-0.5243	-0.5377	-0.5006	-0.5374	**54.4**↑
	GRID	-0.1402	-0.1894	-0.2504	-0.3141	-0.2734	-0.3006	**-0.2447**	
FT5-small	PP	-0.5711	-0.5574	-0.4916	-0.4914	-0.5075	-0.4987	-0.5196	**68.4**↑
	GRID	-0.0518	-0.1449	-0.2106	-0.2129	-0.1517	-0.2396	**-0.1686**	
T5-base	PP	-0.6918	-0.6520	-0.5948	-0.5956	-0.6471	-0.5881	-0.6282	**52.7**↑
	GRID	-0.3060	-0.2404	-0.3549	-0.2893	-0.3439	-0.3259	**-0.3101**	
FT5-base	PP	-0.6092	-0.7222	-0.6488	-0.6273	-0.6631	-0.6225	-0.6489	**58.7**↑
	GRID	-0.2616	-0.3243	-0.3319	-0.3245	-0.3240	-0.2743	**-0.3067**	
T5-large	PP	-0.7275	-0.7625	-0.6137	-0.6257	-0.6351	-0.6416	-0.6677	**51.7**↑
	GRID	-0.3243	-0.3336	-0.3979	-0.3912	-0.2956	-0.3512	**-0.3490**	
FT5-large	PP	-0.7679	-0.7444	-0.6195	-0.6616	-0.7107	-0.6631	-0.6945	**50.4**↑
	GRID	-0.3115	-0.3656	-0.3544	-0.3514	-0.3282	-0.3540	**-0.3442**	

Table 14: Backward Transfer (BWT) comparison between our method and ProgPrompt across multiple task orderings (Orders L1–L6) for T5 and Flan-T5 models. Less negative scores indicate reduced forgetting. Our method consistently improves BWT across all model variants.

k	DBPedia	Amazon	AG News
20	0.5203	0.0000	0.0000
200	0.9674	0.0000	0.8180
1000	0.9880	0.5136	0.8900
2000	0.9892	0.5534	–

Table 15: Accuracy for different datasets with varying sample sizes (k) highlights how model performance scales with more training samples.

0.8114 for Qwen-3-4B). Overall, these results underscore constrained decoding as the primary driver of backward transfer, with gradient-based selection offering complementary scalability.

C.7.1 Impact of Prompt Selection Strategies (Gradient vs. FIFO vs. Random).

As shown in Table 25, our gradient-based method consistently yields the fewest forgotten tasks and achieves competitive or superior BWT across all settings. FIFO occasionally matches our BWT but retains more redundant prompts, resulting in higher forgetting. Random selection performs worst, with unstable BWT and the largest number of forgotten tasks.

C.7.2 Effect of Sample Size and Clustering on Performance .

We empirically find that increasing the number of representative samples beyond 1k per class leads to accuracy saturation with only negligible improvements. This indicates that larger sample sizes may not yield substantial performance gains (Table 15). By contrast, clustering provides both data efficiency and better generalization, highlighting the importance of representative input selection in GRID.

C.7.3 Effect of Prompt Length.

We analyze the impact of prompt length (PL) on performance in Table 16 using the T5-large backbone across task orders L1–L3. We observe a clear trade-off between accuracy, backward transfer (BWT), and forgetting. Short prompts (PL=5) yield competitive accuracy but suffer from increased forgetting, while longer prompts (PL=20) improve BWT (i.e., reduce forgetting) at the cost of degraded accuracy and a higher number of forgotten tasks, likely due to over-parameterization and prompt interference. A moderate prompt length (PL=10) consistently provides the best balance across metrics, achieving strong accuracy with stable BWT and controlled forgetting. Based on these observations, we fix the prompt length to 10 tokens in all experiments.

PL	L1			L2			L3		
	Acc	BWT	FT	Acc	BWT	FT	Acc	BWT	FT
5	78.23	-0.3167	10	81.73	-0.3565	5	84.98	-0.4286	15
10	79.12	-0.3243	11	80.69	-0.3336	5	81.09	-0.3979	18
20	80.58	-0.3396	12	79.37	-0.3159	5	75.55	-0.3206	24

Table 16: Impact of prompt length (PL) on T5-large performance across Order L1–L3. We observe a trade-off: longer prompts improve BWT (less forgetting) but increase the number of forgotten tasks and can degrade accuracy. Optimal performance is generally observed at a moderate prompt length (e.g., 10).

C.7.4 Effect of Queue Size.

Figure 6 shows that moderate queue sizes (e.g., $K{=}5$) provide the best trade-off between accuracy, forward transfer, and backward transfer, while larger queues degrade performance due to redundant and weakly task-relevant prompts.

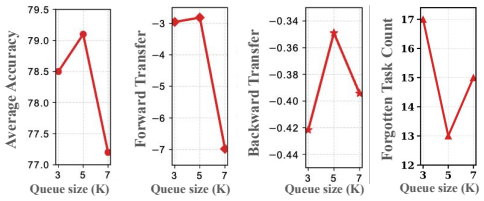


Figure 6: Effect of queue size K.

D Detailed Training and Inference Overhead Analysis.

D.1 Complexity analysis

GRID introduces an additional gradient-scoring step for prompt selection. Importantly, this scoring is performed only once per retained prompt when a new task arrives, rather than during the main prompt-training iterations. For a new task T_t, the scoring overhead is $O(|P_t|)$, where $|P_t|$ denotes the size of the current prompt pool. Across a sequence of N tasks, the total scoring overhead is therefore $O(\sum_{t=1}^{N} |P_t|)$. In practice, GRID maintains a compact prompt pool of approximately five prompts, so the overhead scales nearly linearly with the number of tasks.

D.2 Runtime and Hardware.

In addition to the A100 experiments, we evaluated GRID on H100 GPUs (80GB) with batch size 16. On H100, long task sequences averaged $\sim$ 11 hours per run with peak memory usage of $\sim$ 65 GB, compared to $\sim$ 27 hours and $\sim$ 21 GB on A100 (40GB, batch size 8). These results demonstrate that GRID scales efficiently across hardware generations.

D.3 Training overhead analysis

Table 17 reports the empirical overhead on a 15-task sequence. The gradient-scoring step requires approximately 2.5–3.1 minutes per prompt. Since GRID retains about five prompts, the additional overhead remains around 12.9–15.6 minutes per task, totaling only 144.45 minutes across Tasks 6–15. Overall training time remains comparable to ProgPrompt: GRID requires about 27 hours compared with about 24 hours for ProgPrompt on an A100 with batch size 8. On an H100 with batch size 16, GRID requires about 12 hours compared with about 11 hours for ProgPrompt. These results show that prompt compression introduces only modest training overhead while substantially reducing prompt memory.

Task #	Task	$\lvert P_t \rvert$	min/prompt	Overhead (min)	Cum. (min)
6	boolq	5	2.85	14.25	14.25
7	rte	5	2.92	14.60	28.85
8	imdb	5	2.58	12.90	41.75
9	yelp	5	2.95	14.75	56.50
10	amazon	5	3.11	15.55	72.05
11	sst2	5	2.88	14.40	86.45
12	dbpedia	5	2.90	14.50	100.95
13	agnews	5	2.84	14.20	115.15
14	multirc	5	2.97	14.85	130.00
15	yahoo	5	2.89	14.45	144.45

Table 17: Empirical gradient-scoring overhead for GRID on a 15-task sequence. The scoring step is performed once per retained prompt when a new task arrives. Since GRID maintains a compact prompt pool of approximately five prompts, the additional overhead remains modest.

Method	O1		O2	
	Avg. Acc. ↑	BWT ↑	Avg. Acc. ↑	BWT ↑
ProgPrompt	43.65	-37.27	43.98	-41.65
SHLPT	44.97	-32.20	46.97	-33.97
GRID	**46.54**	**-31.15**	**48.12**	**-31.20**

Table 18: Results on open-ended instruction-based tasks from SuperNI across two task orders. Unlike fixed-label classification tasks, these tasks require open-ended generation. GRID improves both average accuracy and backward transfer (BWT), indicating that its gains are not limited to constrained label-space settings.

D.4 Inference efficiency.

At inference time, GRID uses a substantially smaller prompt pool than prompt-accumulation baselines. For example, after 15 tasks, ProgPrompt retains 15 task-specific prompts, whereas GRID retains approximately five prompts. This reduces the number of prompt tokens prepended to each input. For long inputs with sequence length around 2048, this yields a modest inference speedup of about 5–20%. For shorter inputs with sequence length around 264, where prompts constitute a larger fraction of the total input length, inference becomes about 30–50% faster. Thus, although GRID introduces a small one-time scoring overhead during training, it improves memory efficiency and reduces inference cost throughout deployment.

Model	Setting	L1	L2	L3	L4	L5	L6	Avg
T5-small	PP	63.18	57.18	61.12	62.73	59.11	60.33	60.61
	GRID	59.75	64.97	59.90	64.64	64.51	62.39	**62.69**
FT5-small	PP	60.32	59.25	59.43	59.18	59.35	59.96	59.58
	GRID	58.35	57.37	64.73	61.74	58.30	57.07	59.59
T5-base	PP	71.34	67.60	69.57	69.71	72.43	69.93	70.10
	GRID	72.74	71.24	72.09	67.16	75.72	65.23	70.70
FT5-base	PP	63.73	74.83	75.23	72.14	74.25	72.48	72.11
	GRID	77.24	78.08	77.16	78.32	73.53	65.92	**75.04**
T5-large	PP	75.68	78.56	74.29	75.05	77.10	75.46	76.02
	GRID	79.12	80.69	81.09	79.05	79.79	75.54	**79.21**
FT5-large	PP	79.81	77.51	74.30	79.13	79.09	79.56	**78.23**
	GRID	79.77	78.25	76.09	76.91	76.94	76.74	77.45

Table 19: Extensive comparison of **GRID** (ours) with Progressive Prompts on seq-to-seq backbones (T5 and Flan-T5) across long-sequence task orders (L1–L6), reporting average test accuracy.

Model	Method	NT1	NT2	NT3	Avg
T5-small	PP	-0.5783	-0.4882	-0.5133	-0.5266
	GRID	-0.1957	-0.1945	-0.2043	-0.1982
Flan-T5-small	PP	-0.4771	-0.3924	-0.4097	-0.4264
	GRID	-0.0807	-0.1601	-0.1171	-0.1193
T5-base	PP	-0.5707	-0.5834	-0.5890	-0.5810
	GRID	-0.1817	-0.2103	-0.2314	-0.2078
Flan-T5-base	PP	-0.6719	-0.5335	-0.6116	-0.6057
	GRID	-0.2911	-0.2074	-0.3162	-0.2716
T5-large	PP	-0.7088	-0.6598	-0.6536	-0.6741
	Ours	-0.3154	-0.3658	-0.3077	-0.3296
Flan-T5-large	PP	-0.7156	-0.6628	-0.6886	-0.6890
	GRID	-0.2858	-0.2888	-0.3636	-0.3127

Table 20: Backward Transfer (BWT) on negative transfer benchmarks (NT1–NT3). Less negative is better.

Model	Setting	NT1	NT2	NT3	Avg
T5-small	PP	57.22	55.51	57.79	56.84
	GRID	58.21	55.08	56.44	56.58
FT5-small	PP	51.57	46.02	48.36	48.65
	GRID	50.40	50.54	49.77	50.24
T5-base	PP	61.18	65.75	66.42	64.45
	GRID	63.34	63.46	63.30	63.37
FT5-base	PP	70.81	60.83	67.25	66.30
	GRID	70.76	61.54	70.05	67.45
T5-large	PP	73.57	73.63	74.26	73.82
	GRID	72.29	74.84	73.07	73.40
FT5-large	PP	75.98	73.98	74.84	74.93
	GRID	76.28	71.56	74.72	74.19

Table 21: Average test accuracy on negative transfer task sequences (NT1–NT3) comparing our method with Progressive Prompts. Our method consistently outperforms the baseline across all models, demonstrating better generalization under reduced task similarity.

Model	Set.	L1	L2	L3	L4	L5	L6	Avg
T5s	PP	-5.07	-11.28	-1.48	-6.09	-9.58	-8.63	-7.02
	GRID	-5.82	-0.11	-5.55	-0.96	-1.11	-3.33	-2.81
FT5s	PP	-3.57	-5.26	-4.98	-5.33	-4.92	-4.64	-4.78
	GRID	-2.53	-4.09	3.33	0.25	-3.02	-4.64	-1.78
T5b	PP	-3.02	-7.02	-4.80	-4.55	-1.58	-4.23	-4.20
	GRID	-4.93	-6.68	-5.78	-11.32	-2.21	-13.28	-7.37
FT5b	PP	-13.91	-2.08	-1.48	-4.90	-2.60	-4.53	-4.92
	GRID	4.27	5.10	4.04	5.32	0.18	-7.90	1.84
T5l	PP	-4.14	-1.19	-5.58	-2.66	-2.67	-3.81	-3.34
	GRID	-2.82	-1.20	-0.90	-3.11	-2.20	-6.28	-2.75
FT5l	PP	-0.73	-3.17	-6.65	-1.43	-1.53	-0.99	-2.42
	GRID	2.35	0.75	-1.48	-0.64	-0.60	-0.90	-0.09

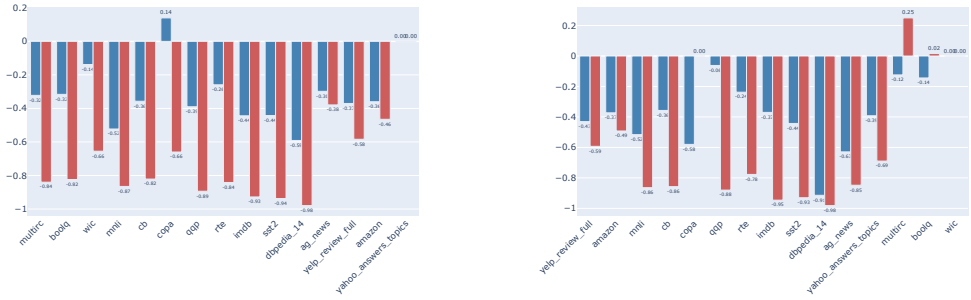
Table 22: Forward Transfer (FWT) scores across various task orderings (Order L1-L6) for T5 and Flan-T5 models. Higher scores indicate better transfer to future tasks. Our method achieves comparable or improved FWT in most cases, demonstrating that reducing forgetting does not come at the cost of forward transfer.

Model	Setting	NT1	NT2	NT3	Avg
T5-small	PP	-5.86	-7.62	-5.62	-6.37
	GRID	-4.67	-7.65	-6.87	-6.40
FT5-small	PP	-3.93	-9.99	-7.88	-7.27
	GRID	-4.96	-4.58	-6.47	-5.34
T5-base	PP	-7.44	-1.35	-1.44	-3.41
	GRID	-6.52	-5.56	-6.64	-6.24
FT5-base	PP	-1.29	-12.68	-5.35	-6.44
	GRID	-1.43	-11.72	-2.05	-5.07
T5-large	PP	-2.55	-2.34	-1.62	-2.17
	GRID	-4.70	-1.85	-3.76	-3.44
FT5-large	PP	0.09	-2.19	-0.76	-0.95
	GRID	1.06	-4.39	-0.77	-1.37

Table 23: Comparison of FWT scores between Progressive Prompts and Ours across different models. The table reports average performance on different tasks order , including Order NT1, NT2, NT3, and the overall average (Avg).

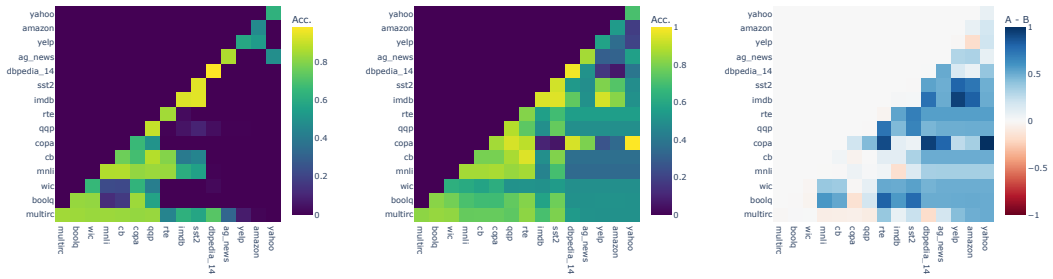
Premise	Choice1	Choice2	Question	Label	ProgPrompt	GRID
The man lost the competition.	The competition was sabotaged.	He intimidated his competitors.	cause	0	false	choice1
I regained composure from my fit of anger.	My heart pounded.	I took deep breaths.	cause	1	true	choice2
The cook's eyes watered.	He ran out of onions.	He cut an onion.	cause	1	true	choice2
The tree branch landed in the river.	The branch moved downstream.	The river's current became stronger.	effect	0	false	choice1
The woman retired.	She received her pension.	She paid off her mortgage.	effect	0	false	choice1

Table 24: Examples highlighting **inconsistent label mappings** in COPA under task-agnostic inference. Prior works like Progressive Prompts use manual conversions (e.g., $0 \rightarrow$ "false"), often misaligning task semantics. GRID mitigates these inconsistencies by inferring appropriate labels automatically.



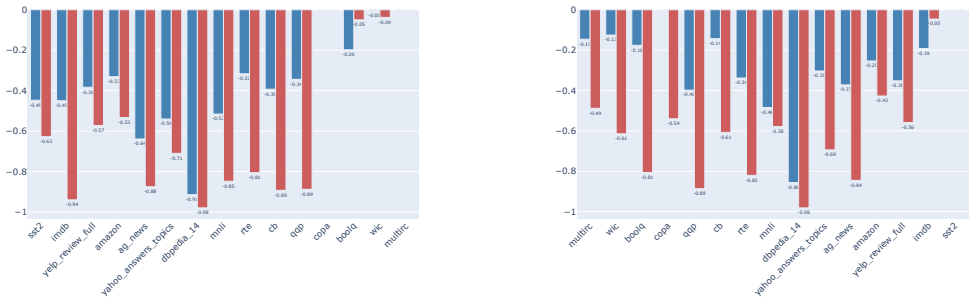
(a) Order L2. (b) Order L3.

Figure 7: Per-task BWT comparison between our method (blue) and the ProgPrompt (red). Positive bars indicate improved retention of prior tasks. Our method shows consistent BWT gains across task orders, demonstrating its effectiveness in mitigating forgetting across diverse task types.



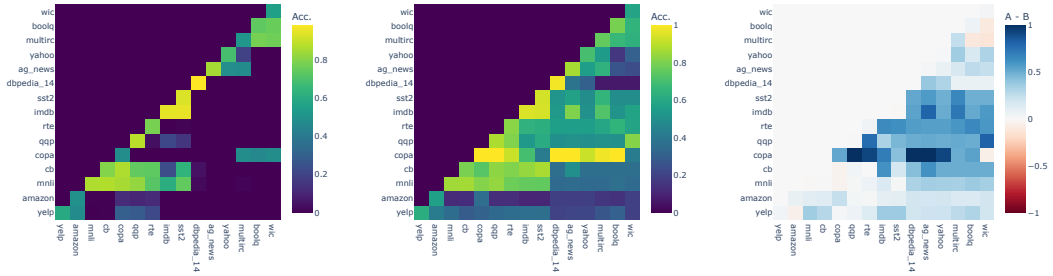
(A) ProgPrompt (B) GRID (C) B-A

Figure 8: Heatmaps of backward transfer scores on previous tasks for Order L2. Brighter values indicate better retention of earlier tasks.



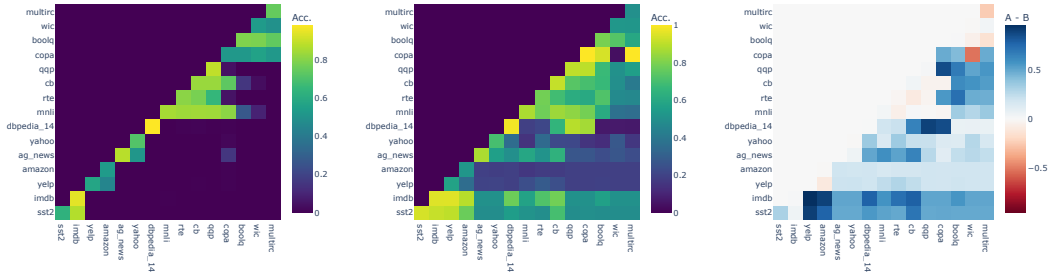
(a) Order L4. (b) Order L5.

Figure 9: Per-task BWT comparison between our method (blue) and the ProgPrompt (red). Positive bars indicate improved retention of prior tasks. Our method shows consistent BWT gains across task orders, demonstrating its effectiveness in mitigating forgetting across diverse task types.



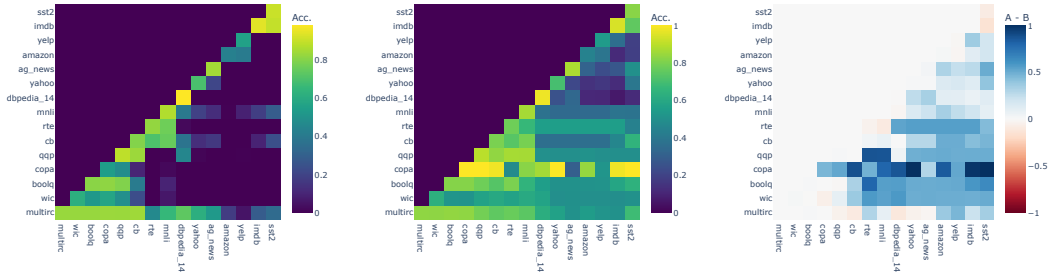
(A) ProgPrompt (B) GRID (C) B–A

Figure 10: Heatmaps of backward transfer scores on previous tasks for Order L3. Brighter values indicate better retention of earlier tasks.



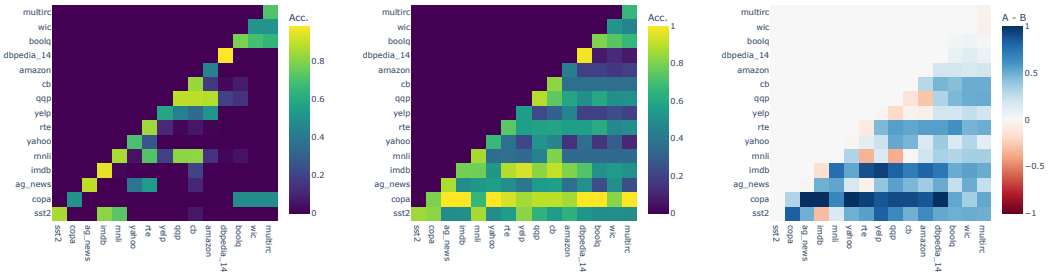
(A) ProgPrompt (B) GRID (C) B–A

Figure 11: Heatmaps of backward transfer scores on previous tasks for Order L4. Brighter values indicate better retention of earlier tasks.



(A) ProgPrompt (B) GRID (C) B–A

Figure 12: Heatmaps of backward transfer scores on previous tasks for Order L5. Brighter values indicate better retention of earlier tasks.



(A) ProgPrompt (B) GRID (C) B–A

Figure 13: Heatmaps of backward transfer scores on previous tasks for Order L6. Brighter values indicate better retention of earlier tasks.

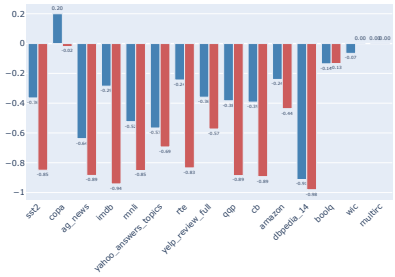


Figure 14: Per-task BWT comparison between our method (blue) and the ProgPrompt (red) for Order L6. Positive bars indicate improved retention of prior tasks.

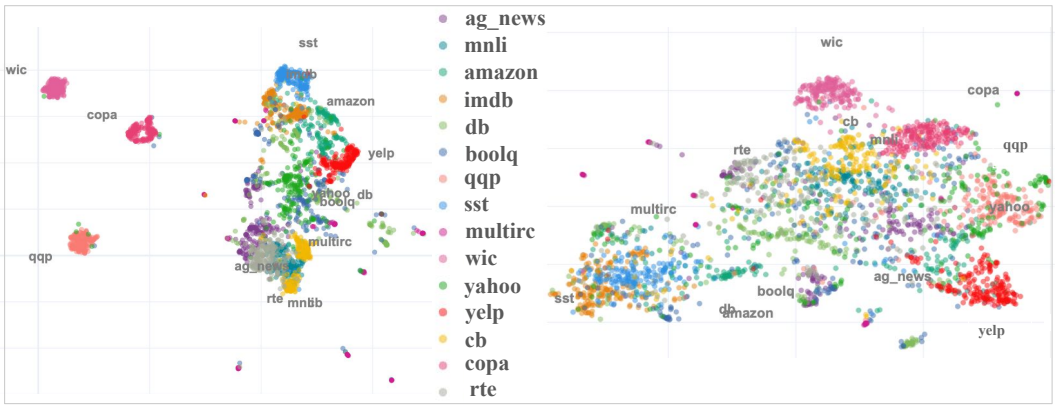


Figure 15: (a) Left - Dataset embeddings using EmbeddingGemma-300M. (b) Right - Dataset embeddings using sentence-transformers/all-MiniLM-L6-v2

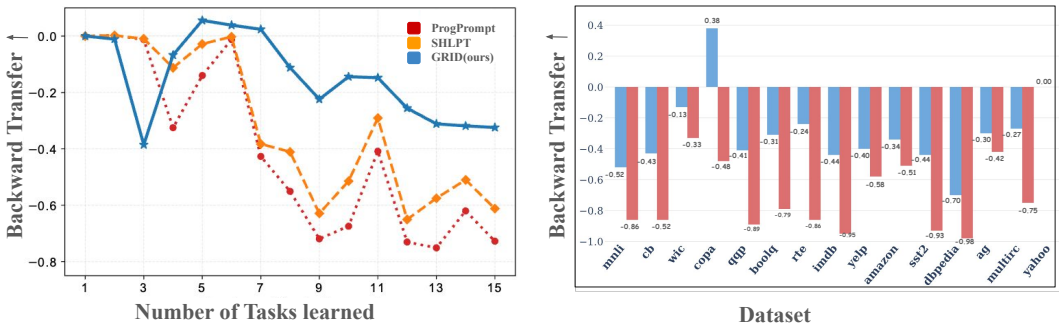


Figure 16: (a) Left - Backward transfer dynamics across sequence (L1). (b) Right - Per-task BWT comparison for Order L1, where GRID (in blue) consistently improves retention over ProgPrompt (in red).

Model	Strategy	L1			L2			L3		
		Acc	BWT	FT	Acc	BWT	FT	Acc	BWT	FT
T5-large	Random	76.81	-0.3482	15	**82.07**	-0.3549	7	**82.06**	-0.4065	21
	FIFO	76.08	-0.3389	14	77.39	-0.3780	11	81.34	**-0.3962**	20
	GRID	**79.12**	**-0.3243**	**11**	80.69	**-0.3336**	**5**	81.09	-0.3979	**18**
Qwen-3-4B	Random	81.75	-0.4767	23	82.15	-0.4835	17	82.28	-0.4705	25
	FIFO	80.62	-0.4781	23	77.05	-0.4981	16	**83.90**	-0.4633	24
	GRID	**83.54**	**-0.4243**	**18**	**82.32**	**-0.4198**	**15**	83.87	**-0.4812**	**23**

Table 25: Comparison of prompt selection strategies (Random, FIFO, Gradient-based) for T5-large and Qwen-3-4B across order L1–L3. Metrics: average accuracy (Acc), BWT, and number of forgotten tasks (FT).

Model	Variant	L1	L2	L3	Avg	Memory	GPU (h:m)
T5-large	**(0) G.R.I.D.**	**−0.3243**	**−0.3310**	**−0.3979**	**−0.3511**	**200**	27:08
	(1) w/o G	−0.3254	−0.3321	−0.3895	−0.3490	600	25:35
	(2) w/o G,D	−0.7032	−0.7589	−0.5967	−0.6863	600	26:42
	(3) w/o G,D,R	−0.7155	−0.7612	−0.5993	−0.6954	600	24:58
	(4) w/o all	−0.7275	−0.7625	−0.6137	−0.7012	600	23:58
Qwen	**(0) G.R.I.D.**	**−0.4243**	**−0.4198**	**−0.4812**	**−0.4418**	**200**	84:72
	(1) w/o G	−0.4145	−0.4210	−0.4823	−0.4393	600	81:38
	(2) w/o G,D	−0.8832	−0.8289	−0.6754	−0.7958	600	81:35
	(3) w/o G,D,R	−0.8855	−0.8297	−0.6876	−0.8009	600	80:46
	(4) w/o all	−0.8987	−0.8421	−0.6934	−0.8114	600	78:47

Table 26: Ablation on GRID over orders L1–L3. We report BWT (less negative is better), average across orders, prompt memory size in KB (slots), and GPU time per run on A100 (40GB) with batch size 8.