HOW MODELS REASON MATTERS: TASK CIRCUIT REPLAY FOR CONTINUAL INSTRUCTION TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mitigating *catastrophic forgetting* remains a major challenge in continual instruction tuning of large language models. Existing methods preserve past knowledge mainly through historical examples, model outputs, or selected internal representations. However, we argue that effective knowledge preservation should go beyond retaining such past information and instead preserve the underlying capabilities acquired from previous tasks. This shifts the focus toward how these capabilities are represented in the model. We argue that how the model computes for a task provides an important characterization of learned capabilities, as task-level computation patterns can consistently support task behavior across different inputs. Based on this perspective, we propose **Task Circuit Replay (TCR)**, a continual instruction-tuning framework that represents and preserves learned capabilities through task-relevant circuits. TCR identifies sparse task-level circuits spanning different model layers, and retains their characteristic responses together with a small number of historical examples. When learning subsequent tasks, these examples reactivate the corresponding circuits, whose responses are selectively preserved to maintain the capabilities they support. To facilitate task-specific circuit identification, TCR introduces Task-BOS to explicitly condition model computation. In this way, TCR preserves past knowledge as reusable task-level computation patterns rather than merely revisiting past observations. Experiments on TRACE and SuperNI demonstrate that TCR consistently reduces forgetting while improving overall performance and backward transfer.

1 INTRODUCTION

Continual instruction tuning enables large language models (LLMs) to continuously acquire new downstream capabilities as tasks and application requirements evolve (Scialom et al., 2022; Shi et al., 2025). However, when multiple tasks are learned sequentially, parameter updates driven by newly introduced tasks can interfere with previously acquired capabilities, leading to substantial performance degradation on earlier tasks. This phenomenon is known as *catastrophic forgetting* (Wang et al., 2023c; 2024a; Shi et al., 2025). Such interference limits the ability of LLMs to accumulate knowledge and skills over time. Therefore, a central challenge in continual instruction tuning is to effectively acquire new capabilities (**plasticity**) while preserving previously learned ones (**stability**) (Qiao et al., 2025).

To mitigate catastrophic forgetting, existing continual learning methods primarily rely on experience replay, regularization, or knowledge distillation (Kirkpatrick et al., 2017; Li & Hoiem, 2018; Rolnick et al., 2019; Buzzega et al., 2020). Recent methods for continual instruction tuning further leverage internal computations in LLMs to identify and preserve task-relevant attention heads (He et al., 2024; Chen et al., 2025), functional representations (Jiang et al., 2024; 2025), or specialized neuron regions (Leng & Xiong, 2025; Chen et al., 2026). By selectively constraining these internal components, these approaches move beyond model-level protection toward more fine-grained preservation of task-specific computations.

Conventional experience replay revisits stored examples, while output-based distillation preserves the model's predictions on selected inputs (Rolnick et al., 2019; Li & Hoiem, 2018; Buzzega et al., 2020). More fine-grained methods further incorporate selected internal representations into knowledge preservation (He et al., 2024; Jiang et al., 2025). However, we argue that effective knowledge

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.

preservation in continual learning should go beyond retaining past examples or outputs to explicitly characterize and preserve the underlying task capabilities, including reasoning abilities and behavioral patterns that extend beyond individual training examples. From this perspective, the central question is *how learned task capabilities should be represented and preserved*. The challenge therefore is to identify a task-level representation of learned capability that captures the internal computation patterns supporting consistent task behavior across different inputs.

To address this challenge, we propose to represent learned task capabilities through task-relevant circuits. In mechanistic interpretability, circuits consist of interacting model components that jointly support specific behaviors (Wang et al., 2023a; Conmy et al., 2023), providing insight into how LLMs represent and use learned knowledge (Yao et al., 2024). We argue that these circuits offer a concrete target for knowledge preservation: the internal computation patterns that support task behavior across different inputs. Building on this representation, we introduce *circuit replay*, which uses historical examples to reactivate task-relevant circuits while preserving the responses of their identified components, with the aim of retaining previously learned capabilities as new tasks are acquired.

Building on this perspective, we propose **Task Circuit Replay (TCR)**, a continual instruction-tuning framework for preserving learned task capabilities through circuit identification and replay. TCR introduces Task-BOS to guide circuit identification with task information, focusing on internal components that consistently contribute to task behavior across examples. The identified circuits then become explicit targets of knowledge preservation during subsequent learning. Rather than treating historical examples only as training data to revisit, TCR uses them to maintain the internal computation supporting previously learned task behaviors. This circuit-level preservation aims to retain the capabilities underlying past experience while allowing the model to adapt to new tasks. Across both model scales, TCR achieves the highest OP in five of the six evaluated settings and remains competitive in the remaining setting. On TRACE, it improves OP over the strongest baseline by 1.65 and 1.25 points on Qwen3-1.7B and Qwen3-8B, respectively.

2 Related Work

In this section, we review the literature most relevant to our work, including continual instruction tuning, mechanistic circuits, and attention sinks in LLMs. We first review existing approaches to mitigating catastrophic forgetting in continual instruction tuning. We then discuss mechanistic circuits and attention sinks, which are closely related to our task-conditioned circuit design.

2.1 Continual Instruction Tuning

Continual instruction tuning aims to continually adapt language models to a sequence of tasks while retaining previously acquired capabilities. Early studies and benchmarks demonstrated both the feasibility of continual instruction learning and the substantial forgetting that arises under sequential adaptation (Yin et al., 2022; Scialom et al., 2022; Zhang et al., 2023; Wang et al., 2023c). Existing approaches primarily mitigate such interference at the data or parameter level. Replay-based methods revisit historical examples or leverage instruction information for more efficient memory use (Yin et al., 2022; Wang et al., 2024b), while parameter-efficient methods isolate task updates through low-rank, selectively routed, or orthogonal parameter subspaces (Wang et al., 2023b; Zhao et al., 2024; Nayak et al., 2026). Recent work further generates task-specific low-rank updates conditioned on task representations (Xiao et al., 2026). While effective, these approaches mainly control which historical information is revisited or which parameters are allowed to change, without explicitly characterizing the internal computation through which previously learned tasks are executed.

Recent evidence suggests that this computational perspective is important for understanding and mitigating forgetting. Fine-tuning can bias a model's implicit task inference (Kotha et al., 2024), while *pseudo forgetting* shows that degraded performance may arise when instructions fail to activate capabilities that remain internally available (Sun et al., 2025). Similarly, function-vector analysis attributes forgetting in part to biased activation of task-processing functions and preserves compact functional representations to improve retention (Todd et al., 2024; Jiang et al., 2025). Other methods move toward finer-grained preservation: SEEKR selectively distills task-sensitive and forgettable at-

tention heads (He et al., 2024), while PaRSP identifies sparse task-specific functional regions and constrains their updates through null-space optimization (Chen et al., 2026). Together, these studies motivate preserving functionally relevant computation rather than uniformly constraining the model. However, their preservation targets are identified from attention sensitivity, functional representations, or parameter specialization, without explicitly distinguishing the causal computation induced by a particular task condition. In contrast, we establish an explicit task-conditioned computational context and preserve the causal components specifically associated with that condition.

2.2 Mechanistic Circuits and Attention Sinks in LLMs

Mechanistic interpretability studies model behavior through the internal computations performed by Transformer components (Elhage et al., 2021). Causal intervention and attribution methods have shown that nontrivial behaviors can often be localized to sparse computational structures spanning attention and MLP components (Wang et al., 2023a; Conmy et al., 2023; Syed et al., 2024; Yao et al., 2024). Recent work further extends this perspective to continual pre-training, revealing that knowledge and concept circuits evolve as information is acquired, interfered with, and forgotten (Ou et al., 2025; Yao et al., 2026). These studies establish sparse internal computation as a meaningful level for analyzing how model capabilities are represented and changed over time. However, existing circuit-based work has primarily used such structures as an *analysis tool*. In contrast, we use causal attribution to identify task-conditioned computational components and subsequently consolidate their historical computation during continual learning.

Our task conditioning further builds on the prominent role of initial tokens in autoregressive Transformers. LLMs commonly exhibit *attention sinks*, whereby early tokens receive disproportionately large attention from subsequent positions (Xiao et al., 2024). Subsequent studies investigate how such sinks emerge and their functional role in Transformer computation (Gu et al., 2025; Barbero et al., 2025). Most closely related to our use of BOS, SinkTrack injects contextual features into the BOS representation and exploits its persistent attention footprint as an information anchor during generation (Liu et al., 2026). While in TCR, rather than using BOS to retain contextual information, we use it as a controllable task-conditioning interface. By contrasting computation under a task-specific BOS condition with a task-agnostic reference, we identify internal components whose causal contributions are specifically associated with the task condition and selectively preserve them during subsequent learning.

3 Method

We consider continual instruction tuning over a sequence of T tasks $\{\mathcal{D}_t\}_{t=1}^{T} = \{(x_i^t, y_i^t)\}_{i=1}^{n_t}$, where x_i^t and y_i^t denote the input instruction and target response respectively. Starting from a pre-trained language model f_{θ_0}, the model sequentially learns $\mathcal{D}_1, \ldots, \mathcal{D}_T$. At each training stage, the model has access to the current task data and a bounded memory constructed from previously observed tasks. The goal of continual instruction tuning is to acquire new task capabilities while mitigating catastrophic forgetting of previously learned tasks.

To preserve historical knowledge at the level of internal computation, we propose a **Task Circuit Replay (TCR)** framework. As illustrated in Fig. 1, TCR comprises three stages. For each task t, we first construct a task-specific condition $\boldsymbol{b}_t$ and inject it into the original *begin-of-sentence* position. This task condition serves as an explicit signal that allows tasks to induce distinguishable internal computation. Based on this condition, we then identify a sparse set of internal components that are specifically modulated by $\boldsymbol{b}_t$ and causally relevant to the corresponding task behavior, thereby isolating task-specific computation from broadly shared language computation. Finally, when learning subsequent tasks, we consolidate the identified historical computation circuits while jointly performing output replay, thus preserving both the external behavior of previous tasks and the internal computation that supports it.

3.1 Conditioning Task-Specific Computation via Task-BOS Injection

Directly identifying task computation from model responses alone may conflate task-specific computation with broadly shared language priors. We therefore introduce an explicit task condition to distinguish task-specific computation. To inject this condition, we leverage the BOS token as a nat-

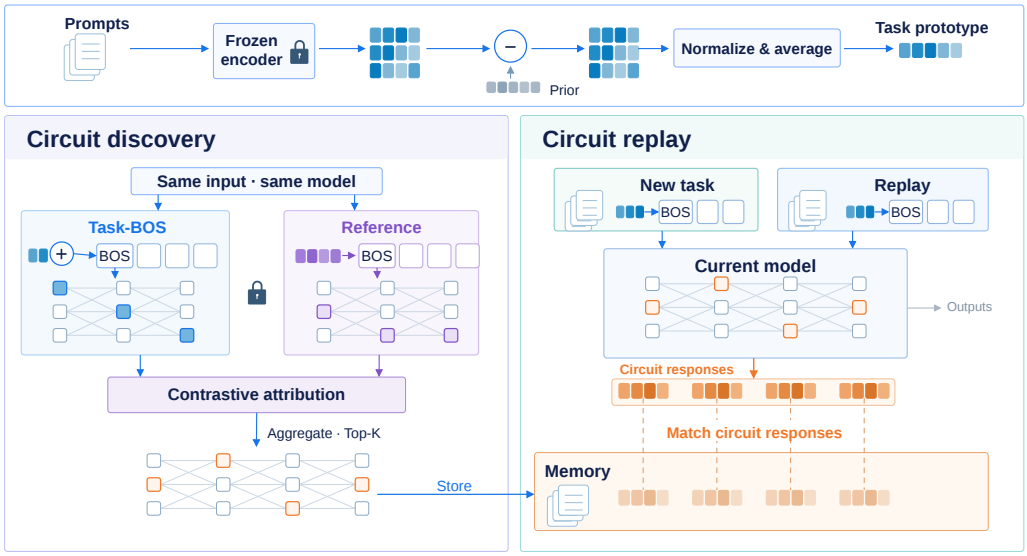


Figure 1: Overview of Task Circuit Replay (TCR). Task prompts are encoded into fixed prototypes and injected at BOS to guide circuit identification. TCR then identifies task-relevant circuit components through contrastive attribution and stores their activations on replay examples as reference targets for subsequent circuit replay. During subsequent training, these examples reactivate the circuit under its original Task-BOS, current activations are matched to stored references alongside example replay and new-task learning.

ural conditioning interface. Under causal attention, the BOS token is accessible to all subsequent tokens, while the attention-sink phenomenon further suggests that early positions can have a persistent influence throughout the forward computation. These properties make the BOS position a suitable entry point for injecting task-specific information, allowing the task condition to consistently modulate downstream computation without altering the original sequence structure.

Task Prototype Construction. We construct the Task-BOS condition using the pre-trained model f_{θ_0}. For an input prompt from task t, let $x_n^t = (x_{n,1}^t, \ldots, x_{n,L_n^t}^t)$ denote its tokenized sequence, where L_n^t is the number of tokens. We use the hidden state of the last prompt token at layer l_p as the sample representation:

$$\mathbf{h}_n^t = \left[\mathbf{H}_{\theta_0}^{(l_p)}(x_n^t)\right]_{L_n^t} \in \mathbb{R}^d. \tag{1}$$

where $\mathbf{H}_\theta^{(l)}(x_n^t) \in \mathbb{R}^{L_n^t \times d}$ is the hidden-state matrix produced at layer l of the model and d is the hidden dimension. However, $\mathbf{h}_n^t$ contains both task-specific information and language priors shared across tasks. To better separate task-specific representations from shared language components, we estimate a fixed language-prior center from an independent language modeling corpus $\mathcal{D}_{\text{prior}} = \{u_j\}_{j=1}^{N_{\text{prior}}}$:

$$\mathbf{c}_{\text{prior}} = \frac{1}{N_{\text{prior}}} \sum_{j=1}^{N_{\text{prior}}} \left[\mathbf{H}_{\theta_0}^{(l_p)}(u_j)\right]_{L_j}, \tag{2}$$

where L_j denotes the number of tokens in u_j. Finally, the task prototype is obtained by normalizing and aggregating the centered representations over task t:

$$\mathbf{b}_t = \text{Normalize}\left(\frac{1}{n_t}\sum_{n=1}^{n_t} \frac{\mathbf{h}_n^t - \mathbf{c}_{\text{prior}}}{\|\mathbf{h}_n^t - \mathbf{c}_{\text{prior}}\|_2}\right). \tag{3}$$

Subtracting $\mathbf{c}_{\text{prior}}$ reduces the contribution of shared language features, while task-level aggregation retains directions that consistently characterize the task across different inputs. Once constructed, $\mathbf{b}_t$ is fixed throughout subsequent continual training. During inference, we retrieve the nearest task

prototype according to the input representation and use it as the task condition. Further details are provided in Appendix A.1.

Task-BOS Injection. Given the task prototype $\boldsymbol{b}_t$, we inject it at the input embedding stage before the first Transformer block. Let $x = (x_0, x_1, \ldots, x_m)$ denote the tokenized input sequence, where x_0 is the original BOS token, and let $\mathbf{H}^{(0)}(x) \in \mathbb{R}^{(m+1)\times d}$ denote the corresponding input representations after token and positional embedding. We construct the task-conditioned input representation as

$$\widetilde{\mathbf{H}}_t^{(0)}(x) = \mathbf{H}^{(0)}(x) + \alpha\, \mathbf{e}_0 \boldsymbol{b}_t^\top, \tag{4}$$

where $\mathbf{e}_0 \in \mathbb{R}^{m+1}$ is the one-hot vector selecting the BOS position and α controls the injection strength. The conditioned representation is then propagated through the original Transformer without any architectural modification. For a pre-normalized Transformer block l, the forward computation is

$$\begin{aligned} \mathbf{U}^{(l)} &= \mathbf{H}^{(l-1)} + \mathrm{Attn}^{(l)}\left(\mathrm{RMSNorm}\left(\mathbf{H}^{(l-1)}\right)\right), \\ \mathbf{H}^{(l)} &= \mathbf{U}^{(l)} + \mathrm{MLP}^{(l)}\left(\mathrm{RMSNorm}\left(\mathbf{U}^{(l)}\right)\right). \end{aligned} \tag{5}$$

with $\mathbf{H}_t^{(0)} = \widetilde{\mathbf{H}}_t^{(0)}(x)$. Thus, Task-BOS does not introduce additional tokens or modify the Transformer architecture. Since BOS token is causally visible to subsequent tokens and exhibits attention-sink behavior, the injected task condition can influence downstream representations through the model's attention and residual pathways.

Simply injecting $\mathbf{b}_t$ does not guarantee that the model actively uses the task condition. During training, we therefore contrast the correct Task-BOS against an irrelevant condition. Specifically, for $(x, y) \sim \mathcal{D}_t$, we define

$$\mathcal{L}_{\mathrm{cond}} = \mathbb{E}_{(x,y)\sim\mathcal{D}_t}\left[\max\left(0,\, \ell(\theta; x, y, \mathbf{b}_t) - \ell(\theta; x, y, \mathbf{b}_{\mathrm{neg}}) + m_{\mathrm{cond}}\right)\right]. \tag{6}$$

Here, $\mathbf{b}_{\mathrm{neg}}$ is sampled from another observed task, while a zero condition is used for the first task. This objective encourages the model to functionally distinguish the correct task condition from irrelevant ones, which provides a meaningful conditional contrast for subsequent circuit identification.

3.2 Identifying Task-Specific Circuits via Contrastive Causal Attribution

After task t has been learned, we identify the internal computation associated with its Task-BOS condition. Our goal is to select components that are *sensitive to the target task condition* and *relevant to the behavior being explained*. This allows us to separate task-conditioned computation from generic computation that would remain active regardless of the task condition.

Let θ_t denote the model parameters immediately after learning task t. We sample a fixed set of anchor examples $\mathcal{A}_t$ and perform circuit identification without further updating the model parameters.

Task-conditioned behavioral contrast attribution. For an anchor example $a = (x, y)$ with target sequence $y = (y_1, \ldots, y_m)$, we use the average teacher-forced target-token log probability as the behavior score:

$$M_{\theta_t}(a, \mathbf{b}) = \frac{1}{m}\sum_{k=1}^{m} \log p_{\theta_t}\left(y_k \mid x, y_{<k}, \mathbf{b}\right). \tag{7}$$

We then perform two forward passes using the same input, target sequence, attention mask, and model parameters. The only difference is the task condition: the first forward uses the correct Task-BOS $\mathbf{b}_t$, while the second uses a reference condition $\mathbf{b}_t^-$. This contrast isolates the behavioral effect associated with changing the task condition. We then attribute this behavioral contrast to individual internal components. Let c denote a candidate internal component, and let $\mathbf{H}_c^+(a)$ and $\mathbf{H}_c^-(a)$ denote its activations under the correct and reference Task-BOS conditions, respectively. We define the contrastive causal attribution of component c as

$$\rho_{t,c}(a) = \frac{1}{L_a}\left\langle \mathbf{H}_c^+(a) - \mathbf{H}_c^-(a),\, \frac{\partial M_{\theta_t}(a, \mathbf{b}_t)}{\partial \mathbf{H}_c^+(a)} \right\rangle, \tag{8}$$

where L_a denotes the length of the tokenized input a. The activation difference measures how strongly the component responds to the Task-BOS condition, while the gradient measures how sensitive the target behavior is to that component. This metric therefore assigns high attribution to components that are both condition-sensitive and behaviorally relevant.

Task-Level Circuit Extraction. Attribution computed from a single example may reflect input-specific variation rather than computation consistently associated with the task. We therefore aggregate the attribution scores over the anchor set $\mathcal{A}_t$. For each candidate component c, we define its task-level importance as

$$S_{t,c} = \frac{1}{|\mathcal{A}_t|} \sum_{a \in \mathcal{A}_t} |\rho_{t,c}(a)| \,. \tag{9}$$

Based on the aggregated scores, we select the K_t components with the largest task-level importance:

$$\mathcal{C}_t = \mathrm{TopK}\left(\{S_{t,c}\}_c, K_t\right). \tag{10}$$

This aggregation produces a sparse task-level computational structure that captures components consistently associated with the Task-BOS-conditioned behavior rather than example-specific activation patterns.

3.3 Consolidating Historical Knowledge via Circuit Replay

Task-conditioned circuit identification provides a sparse description of the computation supporting each learned task. We next preserve this computation when learning new tasks. Instead of constraining the entire hidden representation of the model, we only consolidate the components identified as relevant to historical task behavior.

Circuit Replay Memory. After learning task t, we retain a small replay buffer $\mathcal{B}_t \subset \mathcal{D}_t$. For each replay example $a \in \mathcal{B}_t$ and each identified component $c \in \mathcal{C}_t$, we record its activation under the corresponding Task-BOS:

$$r^{\mathrm{ref}}_{t,c}(a) = h_{\theta_t,c}\left(a, \mathbf{b}_t\right), \tag{11}$$

where θ_t denotes the model immediately after learning task t. We therefore store only a small set of replay examples together with the reference activations of their identified circuit components.

Circuit Consolidation. When learning a later task t, we replay examples from a historical task $i < t$ using its original Task-BOS $\mathbf{b}_i$. We preserve the identified computation by matching the current circuit activations to their stored references:

$$\mathcal{L}_{\mathrm{cir}} = \frac{1}{t-1} \sum_{i=1}^{t-1} \mathbb{E}_{a \sim \mathcal{B}_i} \left[\frac{1}{|\mathcal{C}i|} \sum_{c \in \mathcal{C}i} \left| h_{\theta,c}(a, \mathbf{b}_i) - r^{\mathrm{ref}}_{i,c}(a) \right|_2^2 \right]. \tag{12}$$

Unlike globally matching hidden representations, this objective constrains only the sparse computation identified as relevant to historical tasks, leaving the remaining model computation free to adapt to new tasks.

We additionally perform standard replay on the same historical examples:

$$\mathcal{L}_{\mathrm{out}} = \frac{1}{t-1} \sum_{i=1}^{t-1} \mathbb{E}_{(x,y) \sim \mathcal{B}_i} \left[\ell\left(\theta; x, y, \mathbf{b}_i\right) \right]. \tag{13}$$

Overall Training Objective. When learning task t, the final objective combines current-task learning, Task-BOS conditioning, output replay, and circuit consolidation:

$$\mathcal{L} = \mathcal{L}_{\mathrm{SFT}} + \mathcal{L}_{\mathrm{out}} + \lambda_{\mathrm{cond}} \mathcal{L}_{\mathrm{cond}} + \lambda_{\mathrm{cir}} \mathcal{L}_{\mathrm{cir}}, \tag{14}$$

where λ_{cond} and λ_{cir} control the corresponding auxiliary objectives. For the first task, the historical replay terms are omitted. After learning each task, we identify its task-specific circuit as described in Sec. 3.2 and store the corresponding reference activations for subsequent continual learning.

Table 1: Main results on TRACE, SuperNI-5, and SuperNI-10. Higher OP and BWT are better, while lower FGT is better. Best results are shown in bold and second-best results are underlined.

Method	TRACE			SuperNI-5			SuperNI-10		
	OP↑	BWT↑	FGT↓	OP↑	BWT↑	FGT↓	OP↑	BWT↑	FGT↓
Qwen3-1.7B									
SeqLoRA	41.22	-13.99	13.99	48.63	-1.41	1.41	56.17	-5.99	6.04
ER	<u>55.28</u>	<u>0.80</u>	<u>1.43</u>	49.78	0.24	0.10	60.93	-0.10	1.01
SEEKR	53.11	-0.32	4.14	49.26	0.21	<u>0.07</u>	57.64	-0.07	<u>0.69</u>
FVG	30.47	-8.77	10.29	30.15	-10.38	11.20	25.13	-11.70	14.51
FVG-ER	37.23	-2.07	3.92	40.85	**2.37**	1.40	40.15	**0.42**	5.00
CER	48.70	-6.42	6.69	49.13	-0.51	1.17	57.56	-4.76	4.89
PaRSP	50.00	-5.61	5.79	<u>50.01</u>	-0.08	0.22	<u>61.91</u>	-1.21	1.43
TCR(Ours)	**56.93**	**1.45**	**0.19**	**50.14**	<u>0.42</u>	**0.02**	**62.74**	<u>0.34</u>	**0.47**
Qwen3-8B									
SeqLoRA	52.48	-11.55	11.57	50.68	-1.22	1.40	61.33	-3.52	3.67
ER	60.82	-3.32	3.43	52.09	0.60	0.29	62.85	-1.89	2.39
SEEKR	58.16	-1.23	1.25	51.53	0.14	0.14	62.22	0.09	<u>0.39</u>
FVG	43.57	-1.84	7.91	50.79	0.55	0.27	55.87	-0.62	1.01
FVG-ER	58.07	**12.11**	<u>0.96</u>	51.56	**1.23**	0.10	58.94	**1.00**	1.38
CER	61.01	-2.26	2.86	<u>52.26</u>	<u>0.79</u>	<u>0.07</u>	63.99	-0.99	1.51
PaRSP	<u>64.12</u>	-0.37	1.34	52.15	-0.43	0.43	**65.03**	-0.59	0.72
TCR(Ours)	**65.37**	<u>0.21</u>	**0.62**	**52.29**	0.31	**0.07**	<u>64.67</u>	<u>0.17</u>	**0.28**

4 EXPERIMENTS

In this section, we evaluate the proposed **Task Circuit Replay (TCR)** on continual instruction tuning benchmarks and further analyze its design choices, robustness, and mechanism for preserving task-relevant computation.

4.1 EXPERIMENTAL SETUP

Benchmarks and models. We conduct experiments on TRACE (Wang et al., 2023c) and two continual instruction-tuning sequences constructed from SuperNI (Wang et al., 2022), referred to as SuperNI-5 and SuperNI-10. We evaluate TCR on the **Qwen** (Yang et al., 2025) model family, specifically Qwen3-1.7B and Qwen3-8B, to assess the scalability of the proposed method across different model sizes.

Baselines and evaluation. We compare TCR against SeqLoRA (Hu et al., 2022), standard experience replay (ER) (Rolnick et al., 2019), and recent continual learning methods for LLMs, including SEEKR (He et al., 2024), FVG (Jiang et al., 2025), CER (Yao et al., 2026), and PaRSP (Chen et al., 2026). For replay-based methods, we use the same memory budget of 256 examples per task unless otherwise specified. We evaluate continual learning performance using overall performance (**OP**), backward transfer (**BWT**), and forgetting (**FGT**). Detailed experimental setup, including benchmark descriptions, sequence construction, implementation details, and hyperparameter settings, is provided in Appendix B.

4.2 MAIN RESULTS

As shown in Table 1, TCR achieves the highest OP and lowest FGT across all three benchmarks on Qwen3-1.7B, while maintaining positive BWT. On TRACE, TCR improves OP from 55.28 to 56.93 over the strongest baseline and achieves an FGT of 0.19. Although FVG-ER obtains higher BWT on the two SuperNI sequences, TCR delivers better overall performance with less forgetting. The benefits extend to Qwen3-8B on TRACE, where TCR achieves the highest OP of 65.37 and

Table 2: Ablation and robustness analysis. (a) Component ablations of TCR, where small numbers in parentheses denote OP changes relative to the full model. (b) Comparison between TCR and LRCP under three alternative task orders.

(a) Component Ablation

Method	OP↑	BWT↑	FGT↓
TCR (Full)	**56.93**	1.45	**0.19**
w/o Circuit Replay	56.28 (-0.65)	1.11	0.35
ER-LoRA	55.28 (-1.65)	0.80	1.43
Fixed Random BOS	55.78 (-1.15)	1.51	0.64
Random Circuit	55.95 (-0.98)	0.38	1.02

(b) Task-Order Robustness

Order	Method	OP↑	BWT↑	FGT↓
Order 1	LRCP	58.73	-5.27	5.38
	TCR	**59.17**	**-0.65**	**1.22**
Order 2	LRCP	54.35	-9.90	10.19
	TCR	**58.58**	**-0.94**	**1.79**
Order 3	LRCP	51.73	-11.64	11.72
	TCR	**58.51**	**-0.36**	**1.17**

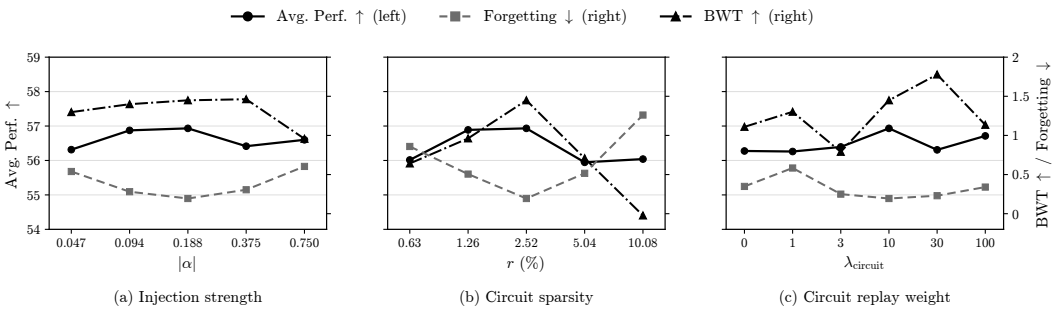
lowest FGT of 0.62, outperforming the strongest baseline by 1.25 points. These results support the effectiveness of circuit replay in preserving learned capabilities during continual instruction tuning.

4.3 Ablation and Robustness Analysis

We further investigate the contribution of the key designs in TCR and assess its robustness to variations in the continual learning sequence. In particular, we conduct component ablations, sensitivity analyses, and evaluations under different task orders. Unless otherwise specified, all experiments are conducted on TRACE using Qwen3-1.7B.

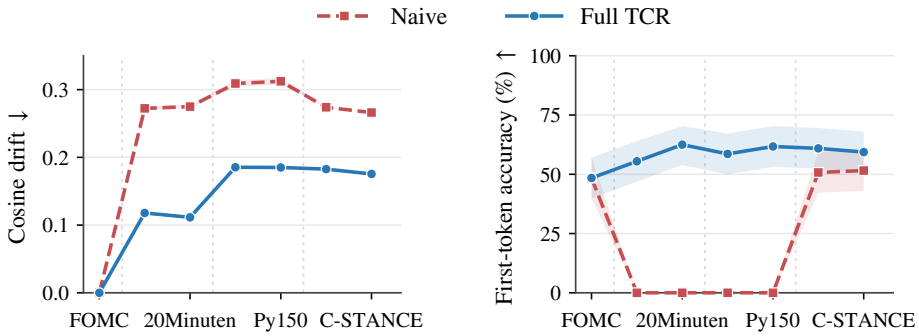
Component ablation. Table 2 (a) shows that full TCR achieves the highest OP and lowest FGT among all variants. Removing circuit replay reduces OP by 0.65 points and increases FGT from 0.19 to 0.35, supporting its contribution to knowledge retention. Replacing the identified circuit with random components increases FGT to 1.02, highlighting the importance of preserving task-relevant rather than arbitrary components. The fixed random BOS variants also yield lower OP and higher FGT, supporting the benefits of task-specific conditioning.

Task-order robustness. Table 2 (b) evaluates TCR under three alternative task orders and compares it with LRCP (Lundstrom-Imanov et al., 2026). TCR consistently achieves higher OP and substantially lower forgetting across all three sequences. In particular, while the FGT of LRCP increases from 5.38 to 11.72 across the alternative orders, TCR maintains FGT within a much narrower range of 1.17–1.79. The advantage in OP also becomes more pronounced under more challenging orders, increasing from 0.44 points on Order 1 to 6.78 points on Order 3. These results suggest that TCR is considerably less sensitive to task ordering and more reliably preserves previously acquired capabilities under different continual learning sequences.



(a) Injection strength (b) Circuit sparsity (c) Circuit replay weight

Figure 2: Sensitivity analysis of TCR. We vary (a) the Task-BOS injection strength $|\alpha|$, (b) the circuit sparsity ratio r, and (c) the circuit replay weight λ_{circuit}. Higher average performance and BWT are better, while lower forgetting is better.



(a) Historical circuit drift

(b) Old-task accuracy

Figure 3: Historical FOMC circuit drift (a) and full-vocabulary first-token accuracy (b). Task-name ticks mark task completion; intermediate points show first-epoch checkpoints. Shading indicates 95% input-bootstrap intervals, not training-seed uncertainty.

Sensitivity analysis. We further study the sensitivity of TCR to three key hyperparameters: the Task-BOS injection strength $|\alpha|$, the circuit sparsity ratio r, and the circuit replay weight λ_{circuit}. As shown in Figure 2, TCR maintains relatively stable performance over a broad range of moderate hyperparameter values. For the injection strength, moderate values yield similar overall performance, whereas an excessively large $|\alpha|$ leads to increased forgetting and reduced backward transfer. For circuit sparsity, performance improves as r increases from very sparse settings to a moderate level, while retaining too many components substantially weakens BWT and increases forgetting, suggesting that selective preservation of task-relevant computation is preferable to broadly constraining internal activations. Finally, TCR remains stable across a wide range of circuit replay weights, with moderate values providing the best trade-off between overall performance and retention, while overly strong circuit regularization begins to reduce BWT. Overall, these results show that TCR does not rely on a narrowly tuned hyperparameter configuration and is robust within reasonable operating ranges.

4.4 Historical Circuit Preservation

We examine whether TCR preserves historical circuit responses as subsequent tasks are learned. As shown in Figure 3(a), full TCR consistently exhibits lower cosine drift than the Naive baseline, reducing final drift from 0.2662 to 0.1755 (34.07%). TCR also maintains old-task first-token accuracy during interference (Figure 3(b)), with a numerically higher final accuracy of 59.38% versus 51.56%. Together, these observations suggest that preserving historical circuit responses helps retain previously acquired task knowledge and mitigate forgetting during continual learning.

5 Conclusion

In this work, we study catastrophic forgetting in continual instruction tuning from the perspective of how previously learned capabilities are internally realized. Rather than preserving historical examples or model outputs alone, we argue that learned task capabilities can be characterized by task-relevant computation patterns that remain consistent across different inputs. Based on this perspective, we introduce **Task Circuit Replay (TCR)**, which uses Task-BOS conditioning to facilitate the identification of sparse task-level circuits and preserves their responses through circuit replay during subsequent learning. Experiments on TRACE and SuperNI demonstrate that TCR improves the retention of previously learned tasks across different model scales and task sequences. Our diagnostic interventions support the causal relevance of adaptation-related components, while the continual-learning analysis shows that TCR better preserves historical circuit responses. These results suggest that preserving *how models compute*, in addition to what they have previously observed, provides a promising direction for continual adaptation of large language models.

AI USE STATEMENT

We used generative AI tools to polish the writing and organization of the manuscript, assist with literature search and reference organization, and support the design and creation of scientific figures. All AI-assisted content was carefully reviewed, verified, and revised by the authors to ensure accuracy, technical correctness, and consistency with the work. We take full responsibility for the final content of this paper, including all text and figures produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or the collection of sensitive data. Our experiments are conducted on publicly available datasets and pretrained language models. We are not aware of any significant ethical risks arising from the methodology or experimental setting of this work.

REPRODUCIBILITY STATEMENT

We provide the methodological and implementation details necessary to reproduce TCR in the main paper and Appendix. In particular, inference-time prototype routing is detailed in Appendix A.1, experimental settings including datasets, baselines, implementation details, circuit configurations, and evaluation protocols are provided in Appendix B, and additional experimental analyses are reported in Appendix C. Our implementation and code will be publicly released after the review process.

REFERENCES

Federico Barbero, Alvaro Arroyo, Xiangming Gu, Christos Perivolaropoulos, Petar Veličković, Razvan Pascanu, and Michael M. Bronstein. Why do LLMs attend to the first token? In *Second Conference on Language Modeling*, 2025.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and SIMONE CALDERARA. Dark experience for general continual learning: a strong, simple baseline. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15920–15930. Curran Associates, Inc., 2020.

Hao Chen, Haoze Li, Zhiqing Xiao, Lirong Gao, Qi Zhang, Xiaomeng Hu, Ningtao Wang, Xing Fu, and Junbo Zhao. ALPS: Attention localization and pruning strategy for efficient adaptation of large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11764–11780, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Jinhui Chen, Shizhu He, Xingchang Yang, Huanxuan Liao, Yequan Wang, Xiangwen Liao, Wenhao Teng, Kang Liu, and Jun Zhao. Harmonizing the past, present, and future: A null-space constrained region-specific method for continual learning in LLMs. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27013–27033, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 16318–16352. Curran Associates, Inc., 2023.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12, 2021.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. When attention sink emerges in language models: An empirical view. In Y. Yue, A. Garg,

N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 97114–97144, 2025.

Jinghan He, Haiyun Guo, Kuan Zhu, Zihan Zhao, Ming Tang, and Jinqiao Wang. SEEKR: Selective attention-guided knowledge retention for continual learning of large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3254–3266, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Gangwei Jiang, Zhaoyi Li, Defu Lian, and Ying Wei. Refine large language model fine-tuning via instruction vector. *arXiv preprint arXiv:2406.12227*, 2024.

Gangwei Jiang, caigao jiang, Zhaoyi Li, Siqiao Xue, JUN ZHOU, Linqi Song, Defu Lian, and Ying Wei. Unlocking the power of function vectors for characterizing and mitigating catastrophic forgetting in continual instruction tuning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 47218–47243, 2025.

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

Suhas Kotha, Jacob Springer, and Aditi Raghunathan. Understanding catastrophic forgetting in language models via implicit inference. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 24110–24139, 2024.

Yongqi Leng and Deyi Xiong. Towards understanding multi-task learning (generalization) of LLMs via detecting and exploring task-specific neurons. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 2969–2987, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

Xu Liu, Guikun Chen, and Wenguan Wang. Sinktrack: Attention sink based context anchoring for large language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Gustav Olaf Yunus Laitinen Lundstrom-Imanov et al. Mechanistic analysis of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2601.18699*, 2026.

Nikhil Shivakumar Nayak, Krishnateja Killamsetty, Ligong Han, Abhishek Bhandwaldar, Prateek Chanda, Kai Xu, Oleg Silkin, Mustafa Eyceoz, Hao Wang, Aldo Pareja, and Akash Srivastava. Sculpting subspaces: Constrained full fine-tuning in llms for continual learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 20212–20243, 2026.

Yixin Ou, Yunzhi Yao, Ningyu Zhang, Hui Jin, Jiacheng Sun, Shumin Deng, Zhenguo Li, and Huajun Chen. How do LLMs acquire new knowledge? a knowledge circuits perspective on continual pre-training. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19889–19913, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Jingyang Qiao, Zhizhong Zhang, Xin Tan, Yanyun Qu, Shouhong Ding, and Yuan Xie. Large continual instruction assistant. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd*

International Conference on Machine Learning, volume 267 of *Proceedings of Machine Learning Research*, pp. 50189–50213. PMLR, 13–19 Jul 2025.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 6107–6122, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. *ACM Comput. Surv.*, 58(5), November 2025. ISSN 0360-0300.

Huashan Sun, Yizhe Yang, Yinghao Li, Jiawei Li, and Yang Gao. Unveiling and addressing pseudo forgetting in large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 23642–23658, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen (eds.), *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 407–416, Miami, Florida, US, November 2024. Association for Computational Linguistics.

Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron Wallace, and David Bau. Function vectors in large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 17282–17333, 2024.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 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*, 46(8):5362–5383, 2024a.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, Singapore, December 2023b. Association for Computational Linguistics.

Xiao Wang, Yuansen Zhang, Tianze Chen, Songyang Gao, Senjie Jin, Xianjun Yang, Zhiheng Xi, Rui Zheng, Yicheng Zou, Tao Gui, et al. Trace: A comprehensive benchmark for continual learning in large language models. *arXiv preprint arXiv:2310.06762*, 2023c.

Yifan Wang, Yafei Liu, Chufan Shi, Haoling Li, Chen Chen, Haonan Lu, and Yujiu Yang. InsCL: A data-efficient continual learning paradigm for fine-tuning large language models with instructions. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 663–677, Mexico City, Mexico, June 2024b. Association for Computational Linguistics.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby

Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

ShiLin Xiao, Tianxiang Xu, Canran Xiao, Weihao Luo, Liwei Hou, and Chuangxin Zhao. Meta-UCF: Unified task-conditioned loRA generation for continual learning in large language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

Barry Menglong Yao, Sha Li, Yunzhi Yao, Minqian Liu, Zaishuo Xia, Qifan Wang, and Lifu Huang. How do large language models learn concepts during continual pre-training? *arXiv preprint arXiv:2601.03570*, 2026.

Yunzhi Yao, Ningyu Zhang, Zekun Xi, Mengru Wang, Ziwen Xu, Shumin Deng, and Huajun Chen. Knowledge circuits in pretrained transformers. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 118571–118602. Curran Associates, Inc., 2024.

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.

Zihan Zhang, Meng Fang, Ling Chen, and Mohammad-Reza Namazi-Rad. CITB: A benchmark for continual instruction tuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9443–9455, Singapore, December 2023. Association for Computational Linguistics.

Weixiang Zhao, Shilong Wang, Yulin Hu, Yanyan Zhao, Bing Qin, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. SAPT: A shared attention framework for parameter-efficient continual learning of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11641–11661, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

A Additional Method Details

A.1 Inference-Time Prototype Routing

At inference time, the task identity of an input is not assumed to be available. We therefore infer the appropriate Task-BOS condition directly from the input prompt by retrieving its nearest task prototype. Suppose that after continual training over T tasks, the prototype bank is

$$\mathcal{B}_T = \{\mathbf{b}_1, \ldots, \mathbf{b}_T\}. \tag{15}$$

Given an unseen input prompt $x = (x_1, \ldots, x_L)$, we first extract its representation using the same frozen pre-trained model f_{θ_0} and the same prototype layer l_p used for constructing the task prototypes:

$$\mathbf{h}(x) = \left[\mathbf{H}_{\theta_0}^{(l_p)}(x)\right]_L. \tag{16}$$

We then remove the shared language-prior component using the fixed prior center $\mathbf{c}_{\text{prior}}$ and normalize the resulting direction:

$$\mathbf{q}(x) = \frac{\mathbf{h}(x) - \mathbf{c}_{\text{prior}}}{|\mathbf{h}(x) - \mathbf{c}_{\text{prior}}|_2}. \tag{17}$$

The resulting vector $\mathbf{q}(x)$ serves as the routing query in the same representation space as the stored task prototypes.

Since each prototype $\mathbf{b}_t$ is normalized, we retrieve the task condition with the largest cosine similarity:

$$\hat{t}(x) = \arg\max_{t \in 1,\dots,T} \mathbf{q}(x)^\top \mathbf{b}_t. \tag{18}$$

Equivalently, because both $\mathbf{q}(x)$ and $\mathbf{b}_t$ have unit norm, this retrieval rule corresponds to nearest-neighbor search under Euclidean distance:

$$\hat{t}(x) = \arg\min_{t \in 1,\dots,T} |\mathbf{q}(x) - \mathbf{b}_t|_2 . \tag{19}$$

The retrieved prototype

$$\mathbf{b}(x) = \mathbf{b}_{\hat{t}(x)} \tag{20}$$

is then used as the Task-BOS condition for the input and is injected through the same Task-BOS injection mechanism used during continual training. The routing operation is performed once from the input prompt before autoregressive decoding, and the retrieved prototype is kept fixed throughout generation.

Importantly, both prototype construction and inference-time routing are performed in the representation space of the frozen pre-trained model f_{θ_0}. Consequently, the routing space remains stationary as the continually trained model evolves, avoiding prototype drift caused by changes in the model parameters across tasks.

B Additional Experimental Details

B.1 Datasets and Task Sequences

TRACE. TRACE (Wang et al., 2023c) comprises eight heterogeneous instruction-tuning tasks covering classification, meeting summarization, code completion, numerical reasoning, and German text simplification. Table 3 summarizes their inputs, outputs, and evaluation metrics. We retain the benchmark-provided training, validation, and test partitions without custom re-splitting. Historical training data become unavailable after their task stage, except for the examples retained in the replay memory.

SuperNI. Super-NaturalInstructions (SuperNI) (Wang et al., 2022) contains 1,616 tasks specified through natural-language instructions. We select ten tasks and construct two continual-learning streams: SuperNI-5 uses the first five tasks, and SuperNI-10 uses all ten. For each task, we sample 2,000 training examples and 500 evaluation examples with seed 42, maintaining disjoint training and evaluation subsets. All selected tasks are evaluated with ROUGE-L F1, including tasks whose outputs are short labels or structured sequences. Table 4 gives the exact task identifiers.

Task sequences. Table 5 lists the default streams and the three TRACE permutations used in the task-order experiment. Each permutation contains the same eight tasks; only their learning order changes.

B.2 Baselines

Sequential adaptation and example replay. SeqLoRA sequentially fine-tunes a single LoRA adapter (Hu et al., 2022) on the current task without replay or an explicit retention objective. ER (Rolnick et al., 2019) revisits stored historical examples during subsequent training, providing a comparison that preserves past data without TCR's task-conditioned circuit selection and activation-matching objective.

Table 3: TRACE tasks and their task-specific evaluation metrics. All metrics are reported on a 0–100 scale, with higher values indicating better performance.

Task	Input and prediction target	Metric
C-STANCE	Chinese microblog text and a target; predict support, opposition, or neutrality toward the target.	Accuracy
FOMC	Statements from Federal Open Market Committee communications; predict a dovish, hawkish, or neutral policy stance.	Accuracy
MeetingBank	City-council meeting transcripts; generate a meeting summary.	ROUGE-L F1
Py150	Python source-code context; generate a code continuation.	Fuzzy accuracy
ScienceQA	A science question with its available textual context and answer choices; predict the correct choice.	Accuracy
NumGLUE-cm	A problem combining commonsense knowledge and arithmetic; generate the numerical answer.	Accuracy
NumGLUE-ds	A problem combining domain-specific knowledge and arithmetic; generate the numerical answer.	Accuracy
20Minuten	A German news article; generate a simplified version.	SARI

Table 4: Selected SuperNI tasks, listed in training order. Each task uses 2,000 training and 500 evaluation examples sampled with seed 42. SuperNI-5 uses rows 1–5; SuperNI-10 uses all rows.

Abbr.	Capability	SuperNI task identifier
QA	Question answering	`task024_cosmosqa_answer_generation`
QG	Question generation	`task074_squad1.1_question_generation`
SA	Sentiment analysis	`task1312_amazonreview_polarity_classification`
Sum.	Summarization	`task511_reddit_tifu_long_text_summarization`
Trans.	Translation	`task1219_ted_translation_en_es`
DSG	Dialogue generation	`task574_air_dialogue_sentence_generation`
Expl.	Explanation generation	`task192_hotpotqa_sentence_generation`
Para.	Paraphrasing	`task177_para-nmt_paraphrasing`
PE	Program execution	`task064_all_elements_except_first_i`
POS	Classification	`task346_hybridqa_classification`

Computation-aware comparisons. SEEKR (He et al., 2024) selectively distills task-relevant attention heads. FVG (Jiang et al., 2025) uses function-vector-based preservation, and FVG-ER combines that approach with historical example replay. CER (Yao et al., 2026) provides a concept-circuit-based comparison. PaRSP (Chen et al., 2026) identifies task-specific functional regions and constrains updates through null-space optimization. Together, these methods compare preservation at the levels of attention heads, functional representations, circuits, and specialized parameter regions. The task-order study additionally includes LRCP (Lundstrom-Imanov et al., 2026), as reported in Table 2.

Comparison protocol. Comparable runs share the backbone, dataset partitions, task order, tokenization, and task-specific evaluation rules. The common optimization settings are given in Table 6; methods retain their own preservation objectives. Replay-based methods use 256 historical examples per task unless otherwise specified. This is an example-count budget: TCR additionally stores task prototypes, component identities, and reference circuit activations, so equal example counts do not imply equal total storage or computation.

B.3 Implementation Details

Backbones and optimization. The main experiments use Qwen3-1.7B and Qwen3-8B (Yang et al., 2025). We freeze the pretrained backbone parameters and train LoRA adapters on the attention and MLP projection layers. Each current task is trained for two epochs with AdamW. The learning-rate schedule restarts at each task, using 3% warmup followed by linear decay. Table 6

Table 5: Default task streams and TRACE orders used for robustness evaluation.

Sequence	Task order
TRACE default	C-STANCE → FOMC → MeetingBank → Py150 → ScienceQA → NumGLUE-cm → NumGLUE-ds → 20Minuten
SuperNI-5	QA → QG → SA → Sum. → Trans.
SuperNI-10	QA → QG → SA → Sum. → Trans. → DSG → Expl. → Para. → PE → POS
TRACE Order 1	FOMC → C-STANCE → ScienceQA → Py150 → MeetingBank → NumGLUE-cm → NumGLUE-ds → 20Minuten
TRACE Order 2	FOMC → C-STANCE → ScienceQA → Py150 → NumGLUE-ds → NumGLUE-cm → MeetingBank → 20Minuten
TRACE Order 3	FOMC → C-STANCE → ScienceQA → Py150 → MeetingBank → NumGLUE-ds → NumGLUE-cm → 20Minuten

Table 6: Training configuration for the continual instruction-tuning experiments. Diagnostic experiments with different exposure or learning rates are specified separately in Appendix C.

Setting	Value
Backbones	Qwen3-1.7B; Qwen3-8B
Trainable parameters	LoRA adapters
Current-task training	Two epochs per task
Optimizer / learning rate	AdamW / 2×10^{-4}
Weight decay	0.01
AdamW $(\beta_1, \beta_2, \epsilon)$	$(0.9, 0.999, 10^{-8})$
Schedule / warmup	Per-task linear decay / 3%
Current-task micro-batch size	2
Gradient accumulation	16 steps
Effective current-task batch size	32
Maximum training sequence length	2,048 tokens
Padding / gradient clipping	Right / 1.0
Numerical precision	BF16 backbone; FP32 LoRA parameters
LoRA rank / scaling / dropout	16 / 32 / 0.05
LoRA target projections	`q, k, v, o, gate, up, down`
Replay example budget	256 per task
Training and data-sampling seed	42

summarizes the shared training configuration. We write r_{LoRA} and α_{LoRA} for adapter rank and scaling, respectively, to distinguish them from the circuit sparsity ratio r and Task-BOS injection strength α.

Task conditioning and historical memory. Task prototypes and the language-prior center are computed with the frozen pretrained model. Once constructed, they remain fixed during continual training. After completing task i, TCR stores its replay examples, Task-BOS vector, selected component set, and the corresponding reference activations. Historical examples are replayed under their original task condition. At stage t, the historical example memory contains at most $256(t-1)$ examples, excluding the current-task data and activation caches. The first task has no historical replay terms; its negative conditioning reference is the zero condition, as described in Section 3.1.

Generation settings. Evaluation uses deterministic greedy decoding with temperature 0, one generated response per prompt, and no beam search. Thinking is disabled; the model's end-of-sequence token determines termination, without custom stop strings. We use a maximum context length of 4,096 tokens. TRACE retains at most 2,048 prompt tokens and generates at most 2,048 new tokens; SuperNI retains at most 3,584 prompt tokens and generates at most 512 new tokens. Prompts exceeding these limits are truncated from the left. For TCR, prototype retrieval precedes decoding, and the retrieved condition is held fixed throughout generation, following Appendix A.1.

Table 7: Parameter values evaluated in the TCR sensitivity study. Circuit sparsity is expressed as a percentage.

Parameter	Values evaluated
Task-BOS strength $\|\alpha\|$	$\{0.047, 0.094, 0.188, 0.375, 0.750\}$
Circuit sparsity r (%)	$\{0.63, 1.26, 2.52, 5.04, 10.08\}$
Circuit-loss weight λ_{cir}	$\{0, 1, 3, 10, 30, 100\}$

Environment and randomness. Experiments use an NVIDIA RTX PRO 6000 Blackwell GPU with Python 3.12, PyTorch 2.11, and CUDA 13.0. The shared seed is 42 for Python, NumPy, PyTorch, CUDA, and data sampling. Repeated discovery-set resampling and bootstrap evaluation in the mechanistic studies quantify the variation specified for those analyses; they are distinct from independent training runs with different initialization or data-order seeds.

B.4 Circuit Discovery and Replay Configuration

Discovery after task learning. TCR identifies the circuit of task i at the checkpoint immediately after learning that task, without further parameter updates. For each anchor, the correct-condition and reference-condition passes share the input, teacher-forced target, attention mask, and model parameters. The behavior score is the mean target-token log probability in Eq. 7. We compute the component attribution using Eq. 8, average its absolute value across anchors, and select the highest-scoring components according to Eq. 10. The selected set is fixed when consolidating that historical task during later training stages.

Reference activations and replay. For every retained example, the reference is the selected component's response at its post-task checkpoint under the original Task-BOS. Later stages compare current responses with these stored references and jointly optimize the circuit and output replay losses in Section 3.3. The circuit objective averages over historical tasks and selected components, while the replayed examples also contribute a supervised response loss. Reference activations serve as fixed targets and are not optimization variables. Only selected components enter the circuit penalty; the model is not globally matched to a historical checkpoint.

Sensitivity settings. The sensitivity study on TRACE with Qwen3-1.7B varies one of the three parameters in Table 7 at a time. The notation λ_{circuit} used in Figure 2 denotes the same circuit-loss weight as λ_{cir} in the training objective. These sweeps concern TCR's conditioning and preservation terms, rather than LoRA rank or adapter scaling.

B.5 Evaluation Metrics

In this section, we introduce the detailed evaluation metrics during continual instruction tuning.

Overall performance and backward transfer. Let $R_{t,i}$ denote the task-specific score on task i after learning task t, where $i \leq t$, and let T be the number of tasks. Evaluation after each task yields a lower-triangular performance matrix over the observed tasks. Overall performance is the unweighted mean of final task scores:

$$\text{OP} = \frac{1}{T} \sum_{i=1}^{T} R_{T,i}. \tag{21}$$

We use *overall performance* rather than average accuracy because the underlying task metrics differ. Backward transfer compares each historical task's final score with its score immediately after acquisition:

$$\text{BWT} = \frac{1}{T-1} \sum_{i=1}^{T-1} \left(R_{T,i} - R_{i,i} \right). \tag{22}$$

Positive BWT indicates net improvement on historical tasks, while negative BWT indicates net interference. Both OP and BWT are higher-is-better.

Forgetting. FGT measures the average non-negative performance drop over the $T-1$ historical tasks:

$$\mathrm{FGT} = \frac{1}{T-1}\sum_{i=1}^{T-1}\max\left\{0,\ \max_{j\in\{i,\ldots,T-1\}} R_{j,i} - R_{T,i}\right\}. \tag{23}$$

For each historical task, we compare its final score with its best score after acquisition and before the final checkpoint. Negative differences are clipped to zero before averaging, so improvements on one task do not offset forgetting on another. The final task is excluded because no subsequent task is learned after it. Lower FGT is better.

B.6 Mechanistic Experiment Setup

Causal interventions and behavioral endpoints. The bidirectional patching study in Appendix C.2 independently adapts Qwen3-0.6B to C-STANCE, FOMC, and ScienceQA. Its diagnostic circuits contain eight attention heads and four MLP modules, selected by LoRA-scale path-integrated attribution and rank consensus over three disjoint discovery folds of 32 examples. Each frozen circuit is evaluated on 256 held-out examples excluded from selection. This diagnostic selection procedure is distinct from the Task-BOS contrast used by the main TCR method. Accordingly, these interventions examine the causal relevance of adaptation-related components rather than providing a direct equivalence between the two selection procedures.

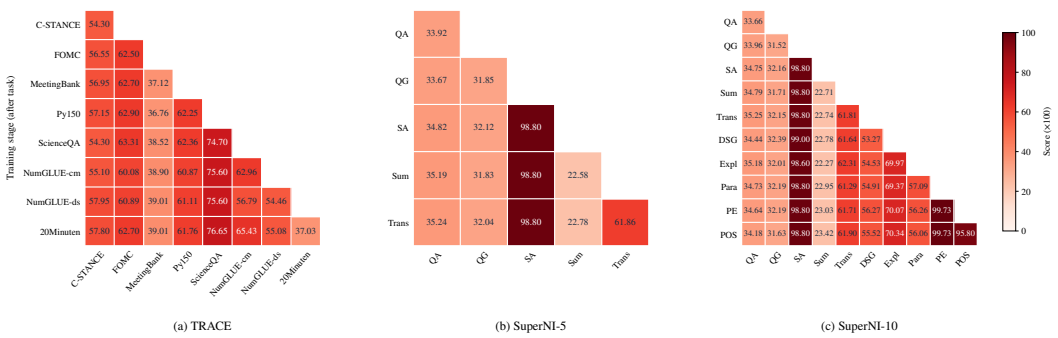
The endpoint is the gold first-answer-label logit minus the log-sum-exp of the alternative label logits. Forward patching inserts adapted-model activations into the original model, and reverse patching performs the opposite replacement. Controls include four component-type- and depth-matched random circuits, position-shifted patches, and self-patches. The effect definitions and results are given in Appendix C.2. This first-label margin is a diagnostic endpoint and is not substituted for the benchmark's free-generation score.

Identification stability and frozen representations. Circuit stability is evaluated on Qwen3-1.7B with five independently resampled discovery sets per task and discovery sizes of 16, 32, 64, 128, and 256 examples. The reported endpoints are node-set Jaccard similarity, Top-K overlap, and Spearman rank correlation of component scores (Appendix C.3). The frozen-representation analysis uses instruction-span pooling over 1,600 TRACE prompts, excluding wrapper, padding, and answer tokens; its fixed-length extraction uses 512 tokens and eager attention. These diagnostic representations differ from the formatted prompt endpoint used by the prototype router. Routing accuracy is evaluated separately on all 7,794 TRACE test inputs, with ground-truth task identities used for scoring (Appendix C.5).

Drift and historical activation restoration. Drift experiments fix the historical circuit and its source activations after learning the old task, then compare subsequent checkpoints with this reference. The retention analysis uses disjoint sets of 128 activation probes and 128 behavioral examples; its retention-error endpoint counts previously correct examples that become incorrect and differs from FGT in Eq. 23. Restoration replaces selected activations in a later checkpoint with source-checkpoint activations for the same input. During free generation, source activations are recomputed under the same evolving prefix as the intervened model. Candidate-label accuracy and free-generation accuracy are reported separately, and paired bootstrap intervals quantify intervention effects. Model sizes, task sequences, exposure settings, and the interpretation limits are specified in Appendices C.6 and C.7.

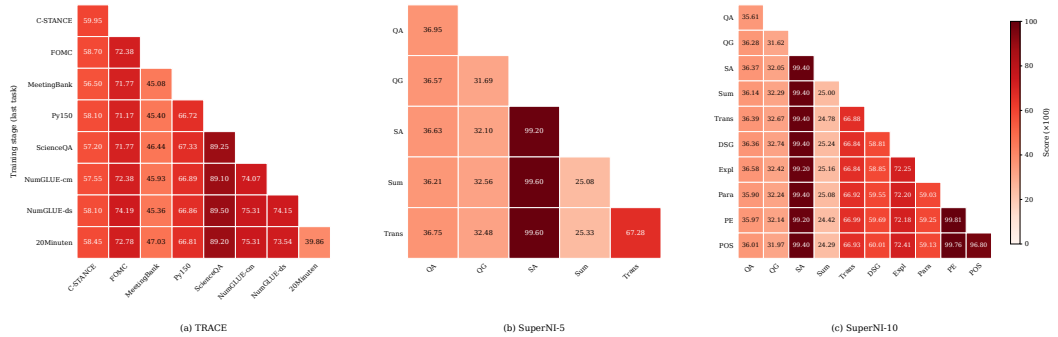
C Additional Experimental Results

This section provides additional experimental results that complement the main evaluation in Section 4. We investigate the properties of the identified task circuits, including their causal relevance and stability under discovery-set resampling. We also analyze the task representations used for Task-BOS construction and inference-time routing, followed by additional diagnostic studies on circuit drift and historical activation restoration. Together, these experiments provide a more comprehensive characterization of both the empirical behavior and the internal mechanisms of TCR.



(a) TRACE (b) SuperNI-5 (c) SuperNI-10

Figure 4: **Task-wise performance trajectories of TCR with Qwen3-1.7B.** Results are shown on (a) TRACE, (b) SuperNI-5, and (c) SuperNI-10. Rows correspond to training stages after sequentially learning each task, and columns correspond to tasks encountered so far. Each cell reports the evaluation score on the corresponding task at a given training stage. Diagonal entries show performance immediately after learning each task, while lower-triangular entries characterize its retention during subsequent training.



(a) TRACE (b) SuperNI-5 (c) SuperNI-10

Figure 5: **Task-wise performance trajectories of TCR with Qwen3-8B.** Results are shown on (a) TRACE, (b) SuperNI-5, and (c) SuperNI-10. The matrix follows the same convention as Figure 4: diagonal entries denote performance immediately after task acquisition, and lower-triangular entries show how previously learned tasks evolve during subsequent continual instruction tuning.

C.1 Task-wise Performance Trajectories Across Model Scales

To complement the aggregate continual learning metrics reported in the main text, we provide a fine-grained view of task-wise performance throughout sequential training for both Qwen3-1.7B and Qwen3-8B. Figures 4 and 5 show the complete performance trajectories on TRACE, SuperNI-5, and SuperNI-10. In each matrix, rows correspond to checkpoints obtained after sequentially learning each task, while columns denote the tasks encountered up to that stage. The diagonal entries represent performance immediately after learning each task, whereas the entries below the diagonal show how the corresponding task performance evolves as subsequent tasks are introduced.

The task-wise results provide a more detailed view of the retention behavior summarized by OP, BWT, and FGT in the main experiments. Across the two model scales, previously acquired task performance generally remains stable as additional tasks are introduced, although the magnitude of the changes varies across individual tasks and benchmarks. The corresponding trajectories also allow us to verify that the aggregate continual learning performance is not driven solely by a small subset of tasks, but reflects retention behavior across the task sequences. Together, these results provide complementary task-level evidence for the effectiveness of TCR across different model scales.

Table 8: Bidirectional causal validation on Qwen3-0.6B. Each task uses a frozen circuit of eight attention heads and four MLP modules, evaluated on 256 held-out examples. Entries are mean first-label logit-margin effects from 10,000 resamples. Panel (a) reports the total adaptation gain and intervention effects; positive reverse removal denotes a margin decrease after replacing adapted activations with original activations. Panel (b) reports paired differences relative to the mean of four type- and depth-matched random circuits and to the position-shifted forward control.

(a) Adaptation and intervention effects			
Task	**Adaptation gain** Δ_{adapt}	**Forward effect** Δ_{fwd}	**Reverse removal** Δ_{rev}
C-STANCE	0.336	0.329	0.277
FOMC	1.526	1.396	1.471
ScienceQA	0.962	1.237	1.005
(b) Intervention effects relative to controls			
Task	**Forward − random**	**Reverse − random**	**Forward − shifted**
C-STANCE	0.642	0.319	0.804
FOMC	2.728	1.028	1.422
ScienceQA	0.607	0.349	1.964

C.2 Causal Validation of Identified Task Circuits

We test whether the identified components causally contribute to adaptation-induced changes in task decisions. This provides a mechanistic check on their functional relevance, motivating their selection as targets for preservation in TCR.

We independently fine-tune Qwen3-0.6B on C-STANCE, FOMC, and ScienceQA using LoRA. For each task, we identify a circuit comprising eight attention heads and four MLP modules using LoRA-scale path-integrated attribution and a rank consensus over three disjoint discovery folds of 32 examples each. We freeze the selected components and evaluate interventions on 256 held-out examples excluded from circuit selection. The endpoint is the first-answer-label margin,

$$m(x) = z_y(x) - \log \sum_{c \in \mathcal{Y} \setminus \{y\}} \exp z_c(x),$$

where $\mathcal{Y}$ is the task's candidate label-token set, y is the gold label, and $z_c(x)$ is the logit for label token c at the first answer position.

We perform bidirectional activation patching between the original model O and its task-adapted counterpart A. Forward patching inserts adapted-model activations into the selected components of the original model. Reverse patching replaces the corresponding adapted-model activations with original-model activations. Writing the resulting models as $O \leftarrow A_C$ and $A \leftarrow O_C$, respectively, we measure

$$\Delta_{\text{adapt}} = \mathbb{E}_x[m_A(x) - m_O(x)], \tag{24}$$

$$\Delta_{\text{fwd}} = \mathbb{E}_x[m_{O \leftarrow A_C}(x) - m_O(x)], \tag{25}$$

$$\Delta_{\text{rev}} = \mathbb{E}_x[m_A(x) - m_{A \leftarrow O_C}(x)]. \tag{26}$$

Positive forward and reverse effects therefore indicate, respectively, transfer and removal of an adaptation-related margin gain. Controls comprise four random circuits matched by component type and depth, a position-shifted forward patch, and self-patches.

Table 8 shows positive forward and reverse effects on all three tasks. Both effects exceed their corresponding matched-random means, and forward patching also outperforms the position-shifted control. Self-patching leaves the measured outputs unchanged. Thus, the selected components causally contribute to the local adaptation-induced decision change, beyond the effects obtained with the matched controls.

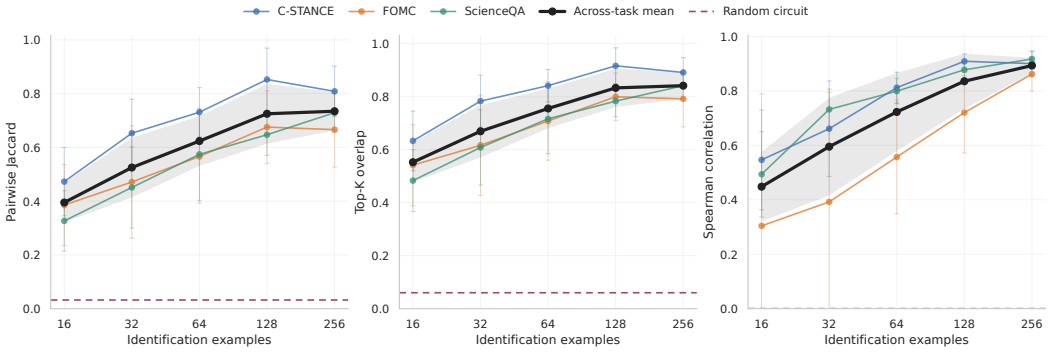


Figure 6: Stability of circuit identification as the number of discovery examples increases. Results are averaged across three classification tasks and five independently resampled discovery sets.

C.3 Stability of Circuit Identification

A useful task circuit should be reproducible under moderate changes to the examples used for circuit discovery. We therefore investigate whether the circuit identification procedure repeatedly recovers similar components when the discovery examples are independently resampled.

Using Qwen3-1.7B, we repeat circuit extraction on three classification tasks with five independently sampled discovery sets for each task. We vary the number of discovery examples and quantify stability using three complementary metrics: node-set Jaccard similarity, Top-K rediscovery rate, and the rank correlation between component attribution scores.

Figure 6 shows that identification becomes substantially more stable as the number of discovery examples increases. The result suggests that the extraction procedure recovers a reproducible task-relevant core rather than an arbitrary subset determined by individual examples. At the same time, the selected component sets are not expected to become perfectly identical, since components close to the selection boundary can change across resampled discovery sets.

C.4 Analysis of Frozen Task Representations

Task-BOS uses representations extracted from a frozen pre-trained model. We examine a prerequisite for this design: whether frozen representations contain signals that distinguish the evaluated tasks, and how those signals depend on layer depth and token aggregation. This analysis concerns per-prompt representations before task-level aggregation or BOS injection.

The instruction span is defined as the complete TRACE prompt field, including both its template and instance content. We extract states from the input embedding layer (L0) and all 28 Transformer layers. Pooling is restricted to retained instruction tokens, excluding chat-wrapper, assistant-prefix, padding, and answer tokens. We compare mean, first-token, last-token, and elementwise-max pooling. For example, the mean-pooled representation at layer ℓ is

$$r_\ell(x) = \frac{1}{|I(x)|} \sum_{t \in I(x)} h_{\ell,t}(x), \tag{27}$$

where $I(x)$ denotes the retained instruction-token positions. Extraction uses a fixed 512-token padded length and eager attention. These settings avoid batch-shape-dependent numerical differences observed in preliminary extraction runs; shared causal-prefix states were checked for exact invariance in the final extraction. The token budget truncates 347 of the 1,600 prompts.

We measure cosine Silhouette in the original representation space and evaluate K-Means clustering with $K = 8$ using normalized mutual information(NMI) and adjusted Rand index (ARI). A linear task-classification probe provides a complementary measure of linear decodability. Its fixed 80/20 split groups examples by exact prompt hash, preventing identical prompts from appearing in both partitions. The held-out partition contains 40 examples per task, or 320 in total; it is drawn from the

sampled TRACE training data, not the official test split. The representative layer is selected by a prespecified rule averaging descending ranks over Silhouette, NMI, ARI, and probe accuracy. Because this descriptive rule uses the reported metrics, the selected-layer scores should not be interpreted as an independent evaluation of layer selection.

Layer-wise separability. Figure 7 shows strong task separation across depth. The aggregate ranking selects L27, where mean pooling achieves Silhouette 0.5861, NMI 0.9880, ARI 0.9860, and probe accuracy 1.0000. The individual metrics do not share a unique optimal layer: Silhouette peaks at L28 (0.6403), NMI and ARI first attain their maxima at L8, and probe accuracy first reaches 1.0000 at L4. Thus, task identity is already readily decodable in early layers, while later contextualization changes the geometry of the representation space. L27 is the representative layer under the joint rule, rather than the unique optimum for every metric.

Aggregation and template-token collapse. Figure 8 compares pooling strategies at L27. Mean pooling achieves higher NMI and ARI than the alternatives while matching the best probe accuracy. First-token pooling has the highest Silhouette (0.7500), but its ARI is 0.8705, its probe accuracy is 0.8750, and K-Means produces only seven non-empty clusters despite requesting eight. NumGLUE-cm and NumGLUE-ds share the initial instruction token `Solve`; its causal representation cannot incorporate the subsequent distinguishing content and was verified to be identical across these tasks. The high Silhouette therefore reflects compact groups that do not distinguish all eight tasks. Last-token pooling reaches perfect probe accuracy but lower NMI (0.9010) and ARI (0.7960), showing that linear decodability need not imply equally strong recovery by K-Means. These comparisons favor mean pooling as a balanced representation for this diagnostic and demonstrate why Silhouette alone is insufficient.

Geometry and qualitative visualization. At L27, mean within-task cosine distance is 0.0172, compared with between-task distance 0.0715, giving a ratio of 0.2408. These original-space measurements support the observed task grouping. Figure 9 provides complementary visualizations at L0, L7, L14, L21, L27, and L28, using PCA to 50 dimensions followed by t-SNE. Task structure

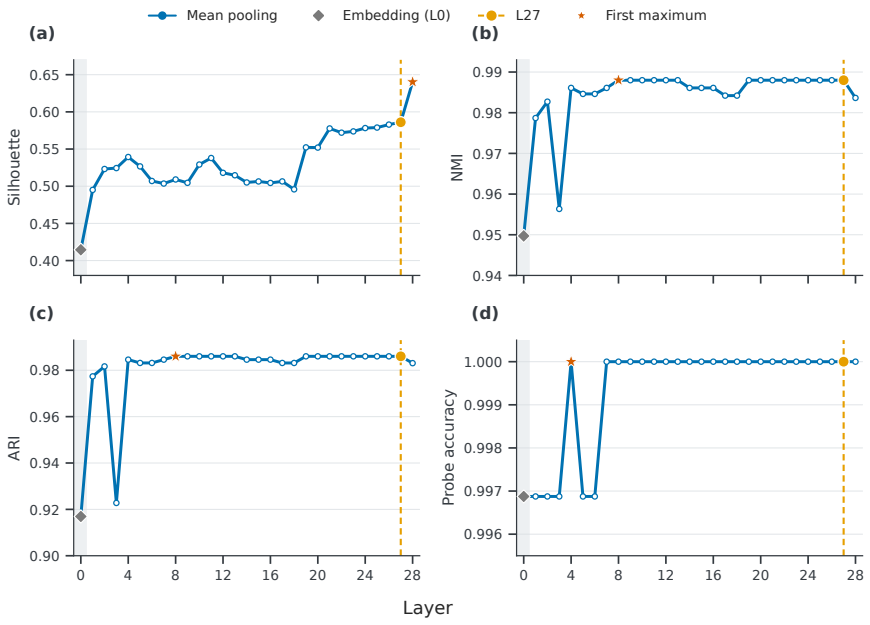


Figure 7: Layer-wise separability of mean-pooled frozen representations: (a) cosine Silhouette, (b) NMI, (c) ARI, and (d) held-out probe accuracy. L0 denotes the input embedding and L1–L28 denote Transformer outputs. Orange markers and dashed lines identify L27; stars mark the first layer attaining each metric's maximum. Vertical scales differ across panels. Results use one fixed sample and seed.

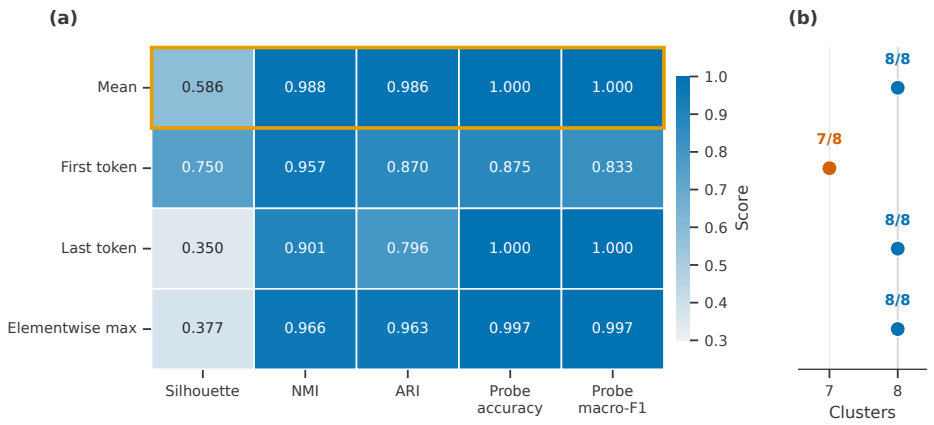


Figure 8: Pooling comparison at L27. (a) Clustering and held-out probe metrics; the outlined row denotes mean pooling. (b) Number of non-empty clusters obtained with K-Means configured for $K = 8$. All strategies use the same retained instruction spans and probe split.

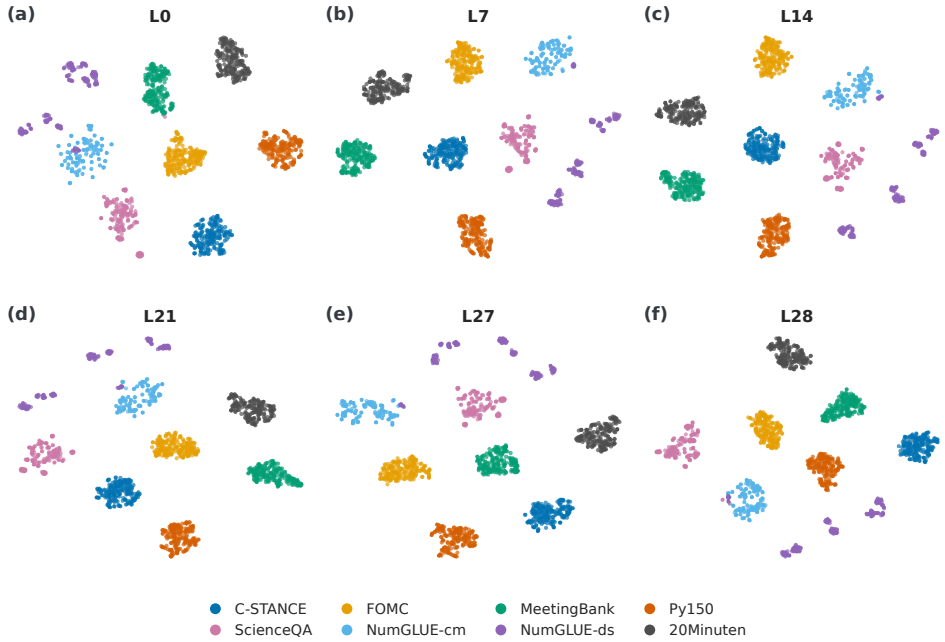


Figure 9: t-SNE views of mean-pooled representations at six layers of frozen Qwen3-1.7B. Each point denotes one prompt; colors identify the eight TRACE tasks. PCA to 50 dimensions precedes t-SNE, with perplexity 30 and seed 42. Panels are separate projections and serve only as qualitative illustrations.

is visible even at L0, consistent with the input-embedding control. The projections are qualitative: their coordinate systems are not aligned across layers, and apparent distances or cluster sizes in two dimensions are not used to quantify separability.

Implications and scope. The evidence supports a specific prerequisite for Task-BOS: frozen representations contain readily accessible task-discriminative signals under the evaluated TRACE prompts. It does not establish abstract, template-independent task semantics, robustness to prompt rewriting, or generalization to unseen tasks. The study uses one sampling seed and one probe split, and exact-hash grouping does not eliminate shared templates or near-duplicate content. Finally, instruction-span pooling here differs from the formatted-prompt endpoint representation used by the

routing implementation. This analysis consequently motivates frozen representations as conditioning candidates; the reliability of the deployed extractor and router requires separate evaluation.

C.5 Evaluation of Prototype Routing

Task-BOS can be selected without observing the input's task identity. As described in Section A.1, the router matches a frozen-model input representation against the stored task prototypes and returns the Task-BOS condition associated with the nearest prototype. We evaluate this procedure as an eight-way task-identification problem on TRACE, with all eight task prototypes available at inference time. Ground-truth task identities are used only for scoring.

We evaluate the final routing procedure on all test examples. Table 9 reports the per-task results. The router achieves 99.87% micro accuracy and 99.64% macro accuracy. Six tasks achieve 100% accuracy; MeetingBank and NumGLUE-ds achieve 99.86% and 97.23%, respectively. The ten errors comprise nine NumGLUE-ds inputs routed to NumGLUE-cm and one MeetingBank input routed to FOMC.

As an exact-duplicate sensitivity check, we exclude inputs matching the prototype-training or external-prior data and deduplicate the remaining test prompts. Accuracy remains 99.88% on the resulting 7,525 inputs. This check addresses exact prompt overlap, rather than near-duplicate content or shared task templates.

These results demonstrate highly accurate Task-BOS selection without oracle task IDs within the eight known TRACE tasks. The endpoint is condition-selection accuracy, not downstream answer accuracy: this experiment does not evaluate the effect of routing errors on generated responses or recognition of previously unseen tasks.

Table 9: Task-BOS prototype routing on the eight TRACE tasks using frozen Qwen3-1.7B. All task prototypes are available during evaluation. Micro accuracy weights examples equally; macro accuracy weights tasks equally.

Task	**Correct / Total**	**Acc. (%)**
C-STANCE	2000 / 2000	100.00
FOMC	496 / 496	100.00
MeetingBank	691 / 692	99.86
Py150	2000 / 2000	100.00
ScienceQA	2000 / 2000	100.00
NumGLUE-cm	81 / 81	100.00
NumGLUE-ds	316 / 325	97.23
20Minuten	200 / 200	100.00
Overall (micro)	7784 / 7794	**99.87**
Task average (macro)	—	**99.64**

C.6 Circuit Drift during Sequential Adaptation

We investigate whether sequential adaptation changes historical task circuits and whether these changes accompany the loss of previously correct predictions. The historical FOMC circuit contains eight attention heads and four MLP modules. Its nodes and source activations are fixed after FOMC adaptation, allowing subsequent checkpoints to be compared against the same reference computation.

Historical circuit drift. Figure 10 follows an FOMC→C-STANCE trajectory across 11 checkpoints. We measure activation changes using cosine distance and normalized squared error on concatenated circuit activations, and separately measure changes in re-identified node membership using $1 - \mathrm{Jaccard}$. At the final checkpoint, these metrics reach 0.180, 0.393, and 0.400, respectively. Activation drift generally accumulates, while node-set changes are non-monotonic. These observations establish that historical circuit states change during subsequent adaptation.

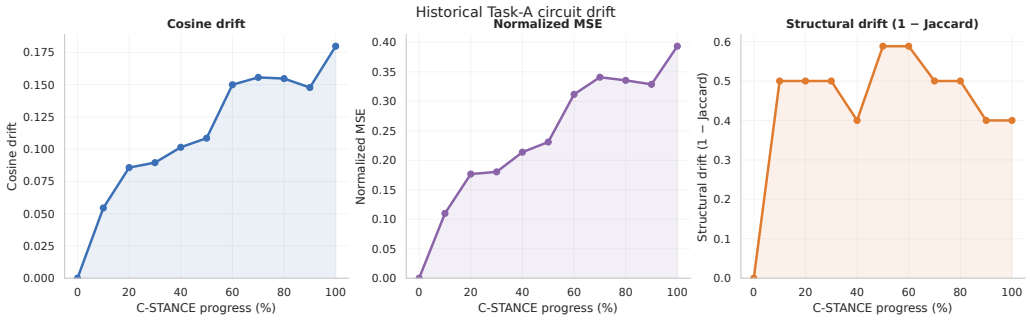


Figure 10: Historical FOMC circuit drift during subsequent C-STANCE adaptation. The three panels report cosine distance and normalized MSE on concatenated activations of the fixed source circuit, and structural drift measured by one minus the Jaccard similarity between the source and re-identified node sets. All 11 checkpoints are shown. Activation drift generally accumulates, whereas structural changes are non-monotonic.

Separating retention loss from compensating gains. Net accuracy decline combines two transitions: previously correct examples becoming incorrect, and previously incorrect examples becoming correct. To isolate the former, our subsequent exploratory analysis measures *retention error* on the fixed set of source-correct examples,

$$S_0^+ = \{i : \hat{y}_i(0) = y_i\}, \qquad F_{\text{ret}}(t) = \frac{1}{|S_0^+|} \sum_{i \in S_0^+} \mathbf{1}[\hat{y}_i(t) \neq y_i].$$

Predictions are evaluated among the FOMC candidate labels. This endpoint quantifies loss of previously correct predictions without offsetting it by gains on initially incorrect examples; it is distinct from net accuracy decline and free-generation answer accuracy.

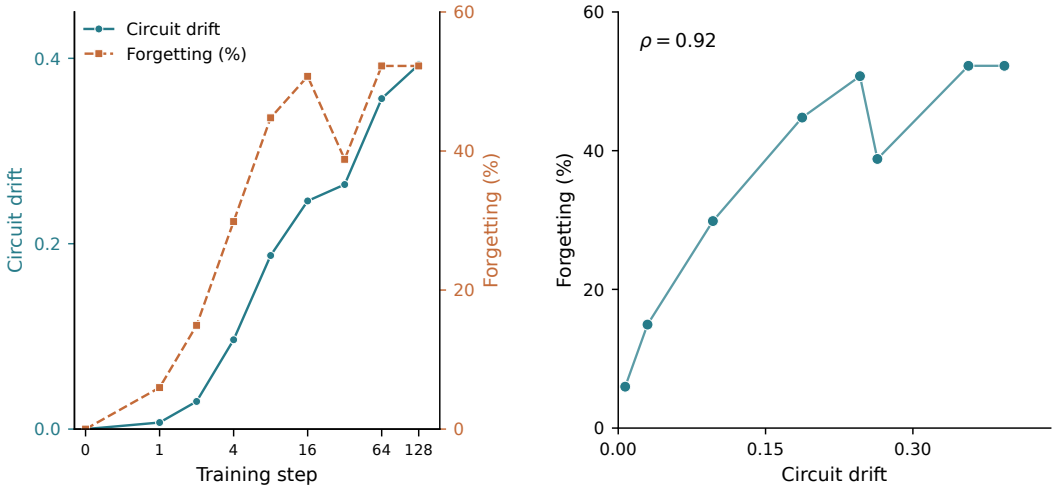
The subsequent stress analysis uses a node-balanced activation distance,

$$D(t) = \frac{1}{|P|\,|C|} \sum_{x \in P} \sum_{v \in C} \left[1 - \cos\left(h_v^{(0)}(x), h_v^{(t)}(x)\right)\right],$$

where C is the fixed source circuit and P is a set of 128 FOMC probes. Unlike the concatenated metric in panel (a), this definition gives each selected node equal weight. Retention is measured on a disjoint 128-example behavioral set, of which 67 examples are correctly classified by the source model.

Association with retention loss. Figures 11 presents an illustrative FOMC→20Minuten stress trajectory using learning rate 10^{-4}. Circuit drift and retention error broadly increase together during adaptation. At the final checkpoint, drift reaches 0.393 and 35 of the 67 source-correct examples are misclassified, giving retention error 52.24%. For comparison, net accuracy decreases by 18.75 percentage points on the complete behavioral set. Across all eight nonzero checkpoints, drift and retention error have Spearman correlation $\rho = 0.922$. The zero-drift source checkpoint is excluded from this calculation. Panels (a) and (b) show the temporal and joint views of the same measurements, rather than independent replications.

Interpretation and scope. The retention analysis was conducted after inspecting the original results, and the displayed 20Minuten trajectory was selected as an illustrative case. Across all four stress branches, cosine drift–retention correlations are 0.826 for C-STANCE at learning rate 2×10^{-5}, -0.515 for C-STANCE at 10^{-4}, 0.627 for Py150, and 0.922 for 20Minuten. Thus, drift is positively associated with retention loss in three of four evaluated trajectories, while the negative branch rules out a universal monotonic relationship. These single-seed, temporally dependent observations do not establish causality or predictive specificity relative to random circuits, and may reflect a common dependence on training progress. They support a limited mechanistic motivation for TCR: historical task-circuit activations can change alongside the loss of previously correct behavior, making their preservation a hypothesis to test through controlled continual-learning experiments.



(a) Evolution during sequential adaptation

(b) Drift–retention association

Figure 11: Circuit drift and retention loss in an exploratory FOMC→20Minuten trajectory. Drift is the node-averaged cosine distance from the fixed post-FOMC circuit, measured on 128 probes. "Forgetting" denotes the fraction of 67 source-correct examples that become incorrect, drawn from a separate 128-example behavioral set. (a) All nine checkpoints, including the source. The curves use separate vertical scales; visual overlap does not imply equal magnitudes. The step axis is logarithmic above step 1 and linear from 0 to 1. (b) The same trajectory plotted in drift–retention space, excluding the source checkpoint, with Spearman $\rho = 0.922$. Lines connect checkpoints in training order without fitting or smoothing. The panels are complementary views of one run, not independent replications. This illustrative case was selected after examining four branches; their correlations are reported in the text.

C.7 Historical Circuit Activation Restoration

We further investigate whether restoring historical circuit computation can recover old-task behavior after subsequent adaptation. We consider the Qwen3-1.7B FOMC→C-STANCE sequence, where the checkpoint immediately after learning FOMC serves as the *source* model and the checkpoint after subsequent C-STANCE adaptation serves as the *final* model. For each FOMC input, we replace the activations of the selected historical circuit components in the final model with their corresponding source-checkpoint activations. The intervention covers eight selected attention heads and four MLP modules. Source activations are computed on the same input, and for free-generation experiments are recomputed using the evolving token prefix of the intervened model. Circuit discovery and confirmation use disjoint examples.

To determine whether the effect is specific to the identified circuit, we additionally compare against matched-random circuits with the same component types and layer distribution, as well as low-score components that receive substantially lower circuit attribution scores. This allows us to distinguish restoration through identified task-relevant computation from generic activation replacement.

As shown in Table 10 and Figure 12, sequential adaptation reduces FOMC candidate-label accuracy from 62.50% at the source checkpoint to 57.42% at the final checkpoint. Restoring the historical activations of the identified circuit raises accuracy to 60.94%, corresponding to a 3.52 percentage-point improvement over the unmodified final model. The intervention therefore recovers a substantial portion of the performance degradation introduced by subsequent adaptation.

More importantly, the effect is specific to the identified historical circuit. The selected intervention outperforms matched-random circuit interventions by 8.01 percentage points and low-score circuit interventions by 10.55 percentage points. It also improves the first-label prediction margin by 0.249 over the unmodified final model. These comparisons indicate that the restoration effect cannot be explained simply by replacing arbitrary internal activations with those from an earlier checkpoint;

Table 10: Historical circuit activation restoration on the Qwen3-1.7B FOMC→C-STANCE sequence. Results are evaluated on 256 confirmation examples excluded from circuit selection. Accuracy values and differences are reported in percentage points.

Evaluation	Result
Source checkpoint accuracy	62.50%
Final checkpoint accuracy	57.42%
Historical circuit restoration accuracy	60.94%
Restoration gain over final	+3.52 pp
Advantage over matched-random circuits	+8.01 pp
Advantage over low-score circuits	+10.55 pp
First-label margin improvement over final	+0.249

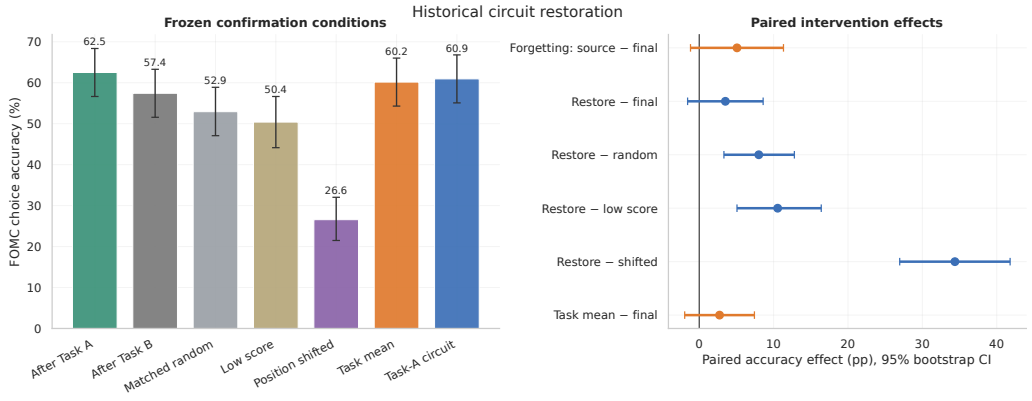


Figure 12: Historical circuit activation restoration after FOMC→C-STANCE sequential adaptation with Qwen3-1.7B. **Left:** candidate-label accuracy at the source checkpoint, after subsequent adaptation, and under historical activation interventions. **Right:** paired intervention effects on 256 confirmation examples excluded from circuit selection. Restoring activations from the identified historical circuit partially recovers old-task performance and yields substantially stronger effects than matched-random and low-score circuit interventions.

instead, components identified as task-relevant carry historical computation that is particularly useful for recovering old-task behavior.

Overall, these results provide causal evidence that the computation carried by the identified historical circuit remains directly relevant to previously learned task behavior. When subsequent adaptation alters this computation, reinstating its historical activation pattern can partially recover old-task performance. Together with the circuit-preservation results, this finding supports the central motivation of TCR: preserving task-relevant circuit responses provides a targeted mechanism for retaining previously acquired capabilities during continual learning.