# Route, Reuse, Repurpose: Continual Adaptation of LLMs with Bounded Adapter Pools

**Bilge Celik** [1] **Anna Vettoruzzo** [1] **Joaquin Vanschoren** [1]

## Abstract

Re-training Large Language Models (LLMs) each time a new task or domain emerges is computationally prohibitive, making continual adaptation a central challenge. We introduce CaLLM, a modular framework featuring metric-based meta-learning and parameter isolation via lightweight adapters with controllable plasticity. By projecting inputs into a task-agnostic embedding space, a non-parametric, task- and regime-agnostic router maps data to its nearest prototype adapter. This design ensures training FLOPs, memory usage, and router overhead remain strictly constant regardless of the adapter pool size. When capacity saturates, CaLLM repurposes the least-used adapter, enabling bounded-cost intentional knowledge removal. We evaluate CaLLM on cross-domain shifts and long-horizon dialogue streams, under both batched and online scenarios, where CaLLM outperforms baselines achieving the largest gains on the most challenging tasks.

## 1. Introduction

Large language models (LLMs) have become remarkable generalists, but real-world deployment often requires them to absorb task-specific expertise without sacrificing broader capabilities. A recent survey (Shi et al., 2025) formalizes this tension along two axes: vertical continuity, from broad pretraining to task-specific fine-tuning, and horizontal continuity, adapting to shifting tasks and data distributions over time. In this research, we address both axes.

Classical continual learning (CL) addresses forgetting through regularization, replay, and parameter isolation (Kirkpatrick et al., 2017; Rolnick et al., 2019; Rusu et al., 2016), but these approaches face scaling trade-offs in LLMs, such as growing memory, costly retraining and poor robustness under domain shift. Recent LLM continual fine-tuning therefore relies on lightweight-adapter methods (Hu et al., 2021) that reduce interference via subspace constraints, learned gating, or MoE routing (Wang et al., 2023a; Liang & Li, 2024; Qiao et al., 2024; Liang & Li, 2025; Wu et al., 2024; Wang et al., 2024; He et al., 2024; Dou et al., 2024; Feng et al., 2024; Zhao et al., 2025). Building on metric-based meta-learning and Prototypical Networks (Snell et al., 2017; Vettoruzzo et al., 2024), CaLLM extends prototypes to task-level adapter routing, yielding a training-free router that preserves MoE-style modularity without learned-gating overhead. More details on related work can be found in Appendix A.

We present CaLLM (Continual Adaptation of LLMs), a modular CL method for LLMs, designed for efficient scaling. Instead of using gradient-based router optimization typical of MoE approaches, CaLLM employs a non-parametric router that assigns incoming batches to the most relevant adapter by comparing frozen-backbone embeddings with stored distributional prototypes. This isolates a single active adapter per forward pass, keeping computational overhead constant. The router dynamically expands the adapter pool for novel data distribution and, at saturation, it deliberately overwrites adapters whose knowledge has gone dormant, turning bounded capacity into an opportunity for selective pruning of outdated skills.

We evaluate CaLLM under both batched and online settings on benchmarks spanning severe domain shifts and long-horizon dialogue adaptation. CaLLM improves over CL baselines, with gains most pronounced on tasks where the pretrained backbone is weakest. We further show that while routing stability and task similarity matter more than simply increasing capacity, moderately sized pools can outperform larger ones with constructive interference of grouping related tasks as implicit replay. Code is available at `https://github.com/Bilgecelik/CaLLM-code`.

## 2. CaLLM: Continual Adaptation of LLMs

CaLLM enables configurable forgetting through prototype-based routing and dynamic task-specific adapters. It maintains a pool of lightweight adapters, each specialized in a

[1]AMOR/e Lab, Eindhoven University of Technology. Correspondence to: Bilge Celik <b.celik@tue.nl>.

Presented at the ICML 2026 Workshop "Continual Adaptation at Scale: Towards Sustainable AI". Copyright 2026 by the author(s).

region of the task embedding space. Incoming data are embedded and routed to the most relevant adapter, while novel data can trigger the creation of a new adapter. This design lets CaLLM reuse knowledge when tasks are similar, isolate learning when tasks differ, and adapt as the data distribution evolves. See Figure 3 in Appendix B for system overview.

### 2.1. Modular Adapter Pool

CaLLM maintains a pool $\mathcal{P} = \{(\ell_i, \mathbf{p}_i, c_i)\}_{i=1}^{|\mathcal{P}|}$ bounded by configurable $P$ adapters. Each adapter $\ell_i$ is paired with a prototype embedding $\mathbf{p}_i \in \mathbb{R}^d$ that summarizes the data distribution it was trained on and a usage counter $c_i$. The framework is backbone- and adapter-agnostic, supporting any Transformer-based LLM and any PEFT module that can be independently instantiated, saved, and loaded (e.g. LoRA (Hu et al., 2021), $(IA)^3$ (Liu et al., 2022), or bottleneck adapters (Houlsby et al., 2019)). In our experiments we use LoRA on every linear projection (see Appendix B.1). Crucially, only the currently selected adapter is loaded at a time, so training and inference FLOPs and GPU memory remain constant regardless of $|\mathcal{P}|$. This preserves the MoE-style modularity without the compute footprint of an active mixture, yielding what we refer to as fixed-cost modularity. See Appendix C.2 for quantitative results.

### 2.2. Non-Parametric Prototype Router

**Embedding extraction.** Given a batch of $n$ prompts, the router runs a single adapter-free forward pass through the backbone, mean-pools the final-layer hidden states over valid tokens, and averages across the batch to obtain a distributional embedding $\mathbf{e}$:

$$\mathbf{e} = \frac{1}{n} \sum_{j=1}^{n} \frac{\sum_{t=1}^{s_j} m_{jt} \cdot \mathbf{h}_{jt}^{(L)}}{\sum_{t=1}^{s_j} m_{jt}}, \tag{1}$$

where $m_{jt} \in \{0, 1\}$ is the attention mask, $\mathbf{h}_{jt}^{(L)}$ is the hidden state at position $t$ of sequence $j$ with length $s_j$ in the final transformer layer $L$. Input sequences are tokenized with truncation at a configurable maximum length. Using the frozen backbone without adapters provides embeddings that work from $t = 0$ and remain robust to drift from adapter training.

**Prototype matching.** The router computes the cosine distance between the new embedding $\mathbf{e}$ and each stored prototype $\mathbf{p}_i$:

$$d_i = 1 - \frac{\mathbf{e} \cdot \mathbf{p}_i}{\|\mathbf{e}\| \, \|\mathbf{p}_i\|}, \quad i = 1, \ldots, |\mathcal{P}|. \tag{2}$$

Let $i^* = \arg\min_i d_i$ denote the index of the closest stored prototype and $d_{\min} = d_{i^*}$ its distance. The adapter selection then follows a three-way decision rule governed by a distance threshold $\eta$:

1. **Reuse** ($d_{\min} \leq \eta$): Load the closest adapter $\ell_{i^*}$, update its prototype via an exponential moving average with equal weighting $\mathbf{p}_{i^*} \leftarrow \frac{1}{2}(\mathbf{p}_{i^*} + \mathbf{e})$, increment its usage counter $c_{i^*}$, and fine-tune its parameters on the incoming batch.
2. **Create** ($d_{\min} > \eta$ and $|\mathcal{P}| < P$): Initialize a new adapter from the base weights, add it to the pool with prototype $\mathbf{p}_{\text{new}} = \mathbf{e}$ and usage counter $c_{\text{new}} = 1$.
3. **Repurpose** ($d_{\min} > \eta$ and $|\mathcal{P}| = P$): Repurpose the least-used adapter slot (by usage count $c_i$), reassign its prototype to $\mathbf{e}$, reset its usage counter, and overwrite its weights during subsequent fine-tuning.

The threshold $\eta$ controls the granularity of the pool: lower values encourage more adapters (finer specialization), while higher values promote sharing across related distributions.

When no fixed threshold is provided, an adaptive threshold is computed as half the minimum pairwise cosine distance between the closest prototype and all others:

$$\eta_{\text{adapt}} = \frac{1}{2} \min_{j \neq i^*} \left( 1 - \frac{\mathbf{p}_{i^*} \cdot \mathbf{p}_j}{\|\mathbf{p}_{i^*}\| \, \|\mathbf{p}_j\|} \right), \tag{3}$$

where $i^*$ is as defined above. This ensures the threshold tightens as the embedding space becomes more densely populated. When only one prototype exists, a default threshold of $0.1$ is used.

### 2.3. Training Protocols and Metrics

Keeping the backbone model weights frozen, the selected adapter is fine-tuned with the standard causal-LM cross-entropy on its assigned data. CaLLM supports two regimes that share this objective but differ in data presentation: Batched (CaLLM-B; Algorithm 1, Appendix C.1), where complete task datasets arrive sequentially and backward transfer is measured against each task's score immediately after its own training; and online (CaLLM-O; Algorithm 2, Appendix C.1), where smaller chunks of data from many tasks are randomly interleaved and a strict test-then-train order ensures each chunk is evaluated before any update. We quantify plasticity and stability with Average Performance and Backward Transfer (BWT) in the batched case, and their streaming counterparts Online Moving Average (OMA) and online BWT in the online setting. (Details of training protocols and exact metric definitions are deferred to Appendix B.)

## 3. Experiments & Results

We focus on three questions: (i) Do long-horizon continual scenarios expose the limits of existing CL recipes for LLMs? We show that sequential finetuning and CL baselines either forget substantially or under-adapt. (ii) Can a

non-parametric router maintain plasticity and stability under truly online streaming conditions? CaLLM's router shows consistent task–expert assignments and steady OMA gains through stream. (iii) How does bounded capacity affect knowledge accumulation and pruning stale knowledge? Capacity yields a non-monotonic trade-off where routing stability plays a significant role, revealing balance between specialization, reuse, and intentional overwriting. Additional experiments and per-task results are in Appendix E.

### 3.1. Experimental Setup

We evaluate CaLLM on two benchmarks chosen to mirror the long-horizon, distribution-shifting workloads a deployed foundation model is likely to face. TRACE-8 (Wang et al., 2023b) covers eight heterogeneous tasks spanning domain QA (ScienceQA, FOMC, MeetingBank), multilingual processing (C-STANCE, 20Minuten), code completion (Py150), and math reasoning (NumGLUE-cm/ds), exposing the model to radical cross-domain shifts. CITB-19 (Zhang et al., 2023) consists of 19 dialogue-centric tasks (state tracking, generation, intent identification) derived from Super-NaturalInstructions (Wang et al., 2022), for stress-testing with long-horizon, fine-grained intra-domain streams. To stabilize fine-tuning, we augment each CITB task with high-quality synthetic instances (see Appendix D.5). Each task is evaluated with its appropriate metric; in label-free streams, the metric is selected automatically by an LLM-based classifier. We test both the batched and online regimes defined in §2.3.

**Baselines and backbone.** We compare CaLLM against five baselines including representative CL strategies: Inference-only (frozen backbone), Sequential Adapter Fine-tuning (Seq-FT), EWC (Kirkpatrick et al., 2017), Experience Replay (ER) (Rolnick et al., 2019), and Orthogonal LoRA (oLoRA) (Wang et al., 2023a). All experiments use `gemma-3-12b-it` as the backbone, selected for its strong zero-shot prior on TRACE-8. All baselines share the same LoRA configuration as CaLLM. Further details on experimental setup can be found in Appendix D.

### 3.2. Performance Across Batched and Online Regimes

We first evaluate CaLLM's end-to-end performance in the standard batched task-incremental setting, where tasks arrive one at a time and the system trains on each complete dataset sequentially. CaLLM-B improves over all CL baselines for both benchmarks on average performance. On TRACE-8, it achieves the highest score on 6 of 8 tasks, with the largest gains on the weakest backbone domains (code completion, science reasoning, and math). On CITB-19, CaLLM-B leads on 12 of 19 tasks and maintains a stable trajectory across the full 19-task sequence while Seq-FT and EWC degrade rapidly (See Appendix E.1 for per-task scores and average performance curves.).

The online regime is a more realistic test of continual adaptation: chunks from many tasks arrive interleaved without explicit boundaries, and every chunk is evaluated before any update to the model. Figure 1 summarises CaLLM-O's behavior on the TRACE stream with $P$=8 pool size.

**Knowledge accumulation.** The OMA curve (Fig. 1a) climbs steadily from $\sim$0.29 to $\sim$0.40 over the 500-chunk stream. After a brief volatile phase where the router is still forming its expert assignments, the trajectory becomes monotonically upward, indicating that new chunks are absorbed without disrupting previously learned ones even in a high task-diversity scenario.

**Expert specialization.** The zoomed heatmap (Fig. 1b) shows that each task is consistently routed to the same expert whenever it reappears in the stream, even though chunks arrive in arbitrary order. The router thus behaves as a distribution-sensitive gate, supplying the implicit task labels that explicit CL methods rely on supervision to obtain.

Appendix E.2 shows the full heatmap and CITB results.

### 3.3. The Role of Pool Capacity

A core design choice in CaLLM is the bounded adapter pool: When capacity is reached, the least-used adapter is repurposed, a built-in mechanism for selectively overwriting stale knowledge while keeping memory constant. Here, we examine how performance varies with pool size $P$. Figure 2 visualizes expert assignments and BWT on TRACE-8 in task-incremental setup as $P$ shrinks from 8 to 2.

**Capacity effect on task-incremental learning**. We identify four routing regimes along the capacity–interference spectrum. At $P = 8$, ideal isolation assigns each task to a unique expert, yielding zero BWT loss and showing that router embedding clusters (Appendix E.4, Figure 6a) translate into correct expert assignment. At $P = 6$, constructive clustering emerges: Semantically related tasks such as *NumGLUE-cm/ds* share an expert, while others remain isolated, so aligned gradients make sharing beneficial rather than destructive. At $P = 4$, destructive overload begins as a "super-expert" absorbs disparate QA, stance, and math tasks, causing conflicting updates and a BWT drop to $-0.19$; however, the router still isolates the distinct *Py150* code-completion task. At $P = 2$, all tasks collapse into one expert, BWT falls to $-0.25$, and catastrophic forgetting becomes unavoidable, exposing the trade-off between memory efficiency and interference prevention.

**Stability over capacity in the online regime.** Repeating the analysis on the 500-chunk stream reveals a non-monotonic capacity–performance relation. On TRACE, $P$=4 achieves the highest final OMA (0.42) and the best BWT, outperform-

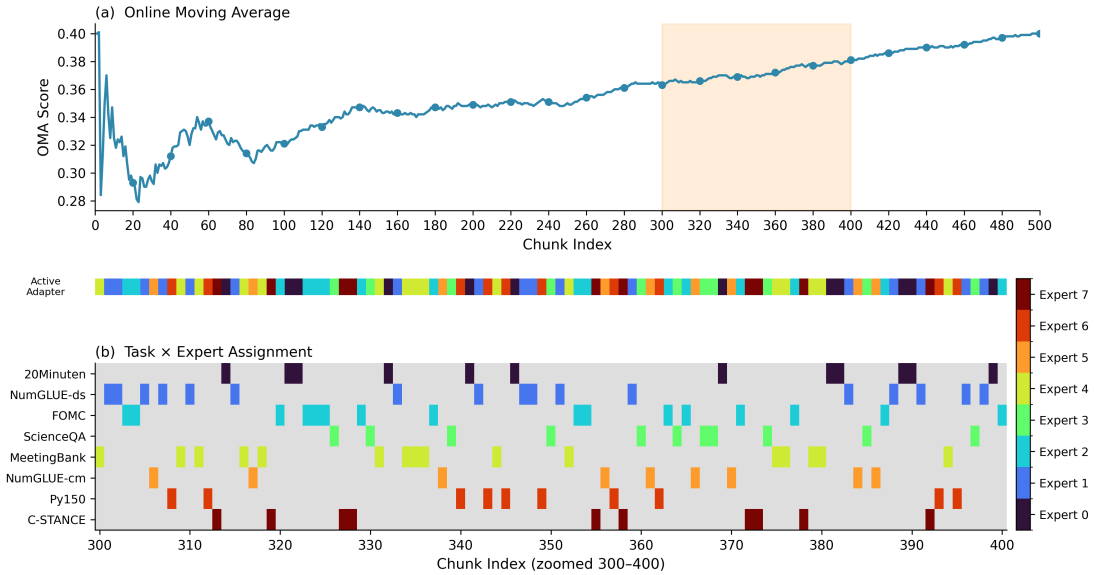



*Figure 1.* **CaLLM-O Online Learning Dynamics (TRACE, $P$=8). (a)** OMA over the 500-chunk stream. **(b)** Task × expert assignment heatmap (chunks 200–300); aggregate adapter timeline on top, per-task routing below. See Appendix E.2 for the full 500-chunk view.

*Figure 2.* **Router behavior under pool capacity constraints (TRACE-8, batched).** Expert-assignment evolution and corresponding BWT curves for $P \in \{8, 6, 4, 2\}$, illustrating the isolation → constructive clustering → destructive overload → collapse progression.

ing both $P$=8 (0.39) and $P$=6 (0.38). We attribute this to routing stability, the consistency of task-to-expert assignment, outweighing the raw capacity (as shown in Appendix E.5.1). At $P$=4, interleaved chunks within each task group on same adapter provide implicit replay via alternating gradient signals, fostering constructive interference. Conversely, $P$=6 suffers from task displacement: 20Minuten is scattered across multiple experts, preventing specialization.

On the more homogeneous CITB-19 stream, performance is far less sensitive to capacity, though smaller pools still produce deeper forgetting spikes (See Appendix E.5 for full plots.).

## 4. Conclusion

We introduced CaLLM, a framework that addresses the resource demands of adapting LLMs to continuously evolving environments. By decoupling a frozen foundational backbone from a bounded pool of lightweight adapters via a non-parametric prototype router, CaLLM mitigates catastrophic interference during unstructured, long-horizon shifts at a constant computational overhead. Crucially, it leverages its capacity limits as an active mechanism for the targeted overwriting of dormant modules.

We view CaLLM as a step toward foundation models that can be updated continuously and sustainably after deployment. Future directions include integrating online AutoML as the stream unfolds, merging adapters to consolidate shared knowledge, and extending prototype-based routing to multi modal streams (see Appendix F for open questions).

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning, specifically the sustainable continual adaptation of foundation models. By keeping training and inference cost independent of the number of accumulated tasks, we aim to reduce the energy and compute footprint typically associated with retraining or scaling large language models, supporting the broader goal of more sustainable AI deployment. The repurposing mechanism additionally provides a controllable handle for selective knowledge removal, which may be useful for privacy, safety, or fact-updating workflows, but it also implies that previously acquired skills can be intentionally overwritten; care should therefore be taken when deploying CaLLM in settings where retention of specific past behaviors is required for compliance or safety. Beyond these considerations, we do not foresee societal consequences of our work that warrant additional discussion here.

## References


Cai, Z., Sener, O., and Koltun, V. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8281–8290, 2021.

De Lange, M. and Tuytelaars, T. Continual prototype evolution: Learning online from non-stationary data streams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8250–8259, 2021.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2021.

De Lange, M., van de Ven, G. M., and Tuytelaars, T. Continual evaluation for lifelong learning: Identifying the stability gap. In *International Conference on Learning Representations (ICLR)*, 2023.

Dikkala, N., Ghosh, N., Meka, R., Panigrahy, R., Vyas, N., and Wang, X. On the benefits of learning to route in mixture-of-experts models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9376–9396, 2023.

Dou, S., Zhou, E., Liu, Y., Gao, S., Shen, W., Xiong, L., Zhou, Y., Wang, X., Xi, Z., Fan, X., Pu, S., Zhu, J., Zheng, R., Gui, T., Zhang, Q., and Huang, X. LoRAMoE: Alleviating world knowledge forgetting in large language models via MoE-style plugin. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1932–1945, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Farajtabar, M., Azizan, N., Mott, A., and Li, A. Orthogonal gradient descent for continual learning. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–40, 2022.

Feng, W., Hao, C., Zhang, Y., Han, Y., and Wang, H. Mixture-of-LoRAs: An efficient multitask tuning for large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING)*, pp. 11371–11380, 2024.

Gutiérrez, B. J., Shu, Y., Qi, W., Zhou, S., and Su, Y. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Han, D., Han, M., and Unsloth team. Unsloth. https://github.com/unslothai/unsloth, 2023.

He, J., Guo, H., Zhu, K., Zhao, Z., Tang, M., and Wang, J. SEEKR: Selective attention-guided knowledge retention for continual learning of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3254–3266, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Huang, J., Cui, L., Wang, A., Yang, C., Liao, X., Song, L., Yao, J., and Su, J. Mitigating catastrophic forgetting in large language models with self-synthesized rehearsal. *arXiv preprint arXiv:2403.01244*, 2024.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Kang, H., Yoon, J., Madjid, S. R., Hwang, S. J., and Yoo, C. D. Forget-free continual learning with soft-winning subnetworks. *arXiv preprint arXiv:2303.14962*, 2023.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Liang, Y.-S. and Li, W.-J. InfLoRA: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23638–23647, 2024.

Liang, Y.-S. and Li, W.-J. Gated integration of low-rank adaptation for continual learning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Liu, H., Tam, D., Muqeeth, M., Phang, J., Tian, L., Raffel, C., and Bansal, M. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Ma, X., Tang, Z., Chen, Z., Peng, C., and Clifton, L. Spurious forgetting in continual learning of language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Qiao, F., Liu, G., et al. Learn more, but bother less: Parameter efficient continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. Experience replay for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Scialom, T., Chakrabarty, T., and Muresan, S. Fine-tuned language models are continual learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

Shenaj, D., Bohdal, O., Ceritli, T., Ozay, M., Zanuttigh, P., and Michieli, U. K-merge: Online continual merging of adapters for on-device large language models. *arXiv preprint arXiv:2510.13537*, 2025.

Shi, H., Xu, Z., Wang, H., Qin, W., Wang, W., Wang, Y., Wang, Z., Ebrahimi, S., and Wang, H. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 2025.

Shi, W., Ajith, A., Xia, M., Huang, Y., Liu, D., Blevins, T., Chen, D., and Zettlemoyer, L. Detecting pretraining data from large language models. *arXiv preprint arXiv:2310.16789*, 2023.

Snell, J., Swersky, K., and Zemel, R. S. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4077–4087, 2017.

Vettoruzzo, A., Bouguelia, M.-R., Vanschoren, J., Rögnvaldsson, T., and Santosh, K. C. Advances and challenges in meta-learning: A technical review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(7):4763–4779, 2024.

Wang, X., Chen, T., Ge, Q., Xia, H., Bao, R., Zheng, R., Zhang, Q., Gui, T., and Huang, X. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, Singapore, December 2023a. Association for Computational Linguistics.

Wang, X., Zhang, Y., Chen, T., Gao, S., Jin, S., Yang, X., Xi, Z., Zheng, R., Zou, Y., Gui, T., Zhang, Q., and Huang, X. TRACE: A comprehensive benchmark for continual learning in large language models. *arXiv preprint arXiv:2310.06762*, 2023b.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A. S., Naik, A., Stap, D., Pathak, E., Karamanolakis, G., Lai, H. G., Purohit, I., Mondal, I., Anderson, J., Kuznia, K., Doshi, K., Patel, M., Pal, K. K., Moradshahi, M., Parmar, M., Purohit, M., Varshney, N., Kaza, P. R., Verma, P., Puri, R. S., Karia, R., Sampat, S. K., Doshi, S., Mishra, S., Reddy, S., Patro, S., Dixit, T., Shen, X., Baral, C., Choi, Y., Smith, N. A., Hajishirzi, H., and Khashabi, D. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. *arXiv preprint arXiv:2204.07705*, 2022.

Wang, Y., Liu, Y., Shi, C., Li, H., Chen, C., Lu, H., and Yang, Y. InsCL: A data-efficient continual learning paradigm for fine-tuning large language models with instructions. *arXiv preprint arXiv:2403.14555*, 2024.

Wang, Z., Li, Y., Qu, X., and Cheng, Y. SEE: Continual fine-tuning with sequential ensemble of experts. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 7418–7432. Association for Computational Linguistics, 2025.

Wu, X., Hartman, M., Jayaraman, V. A., and Varshney, L. R. SwitchCIT: Switching for continual instruction tuning of large language models. *arXiv preprint arXiv:2407.11780*, 2024.

Wu, Y., Wang, J., Chen, X., Zhang, E., Liu, H., Tan, Y., and Li, Y. Exploiting task relationships in continual learning via transferability-aware task embeddings. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yang, S., Ali, M. A., Wang, C.-L., Hu, L., and Wang, D. MoRAL: MoE augmented LoRA for LLMs' lifelong learning. *arXiv preprint arXiv:2402.11260*, 2024.

Zhang, Z., Fang, M., Chen, L., and Namazi-Rad, M.-R. CITB: A benchmark for continual instruction tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9443–9455, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.findings-emnlp.633/.

Zhao, H., Zhu, F., Wang, R., Meng, G., and Zhang, Z. MLLM-CL: Continual learning for multimodal large language models. *arXiv preprint arXiv:2506.05453*, 2025.

Zhu, K., Cao, Y., Zhai, W., Cheng, J., and Zha, Z.-J. Self-promoted prototype refinement for few-shot class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6801–6810, 2021.

## A. Related Work

**Continual learning foundations.** Continual learning aims to accumulate knowledge from non-stationary streams without catastrophic forgetting (De Lange et al., 2021). Existing methods generally utilize replay (Rolnick et al., 2019; Lopez-Paz & Ranzato, 2017), regularization (Farajtabar et al., 2020), or parameter isolation (Rusu et al., 2016; Kang et al., 2023). Beyond architectural design, evaluation is a critical aspect. Recent studies advocate for per-iteration metrics to detect the "stability gap"—a transient but severe forgetting phase during training (De Lange et al., 2023)—and transferability-aware embeddings to leverage inter-task relationships (Wu et al., 2025). Online Continual Learning (OCL) (Cai et al., 2021) further emphasizes learning efficacy via a single-pass "test-then-train" protocol. This regime requires optimizing for immediate adaptation rather than long-term convergence, tracking performance through temporal moving averages instead of static end-of-task scalars.

**Continual learning for LLMs.** Adapting LLMs involves navigating vertical (pretrain-to-fine-tune) and horizontal (temporal shift) continuity. We focus on Continual Fine-Tuning (CFT), evaluated via benchmarks like TRACE (cross-domain robustness) and CITB (long-horizon instruction streams). LoRA is the primary vehicle for CFT, with methods managing interference via subspace isolation, enforcing orthogonality through geometric constraints (Wang et al., 2023a; Liang & Li, 2024) or gradient projection (Qiao et al., 2024). However, these often sum branches with equal weight at inference, failing to select the most relevant parameters. While GainLoRA (Liang & Li, 2025) and SwitchCIT (Wu et al., 2024) introduce gating or switch networks, they require specialized optimization. Conversely, replay-based methods (Scialom et al., 2022; Wang et al., 2024; He et al., 2024) maintain performance but add memory overhead. Notably, recent work suggests that much degradation in LLMs stems from disrupted task alignment rather than true knowledge loss (Ma et al., 2025). Alternatively, non-parametric text-knowledge approaches like Retrieval-Augmented Generation (RAG) bypass parametric updating entirely to avoid forgetting (Gutiérrez et al., 2025). However, these are restricted to retrieving explicit facts rather than adapting procedural or structural capabilities.

**Routing and mixture-of-experts approaches.** Sparse Mixture-of-Experts (MoE) architectures decouple model capacity from per-input computation by routing each token to a small subset of parallel experts (Fedus et al., 2022; Jiang et al., 2024). Learned routers discover interpretable data partitions aligned with latent clusters (Dikkala et al., 2023). In parameter-efficient fine-tuning (PEFT), LoRAMoE (Dou et al., 2024) and MoRAL (Yang et al., 2024) use balancing losses to preserve world knowledge, while MoA (Feng et al., 2024), SEE (Wang et al., 2025), and MLLM-CL (Zhao et al., 2025) assign specific LoRA modules to different tasks or modalities, routing them at inference time to minimize interference. Despite their modularity, these methods generally rely on gradient-based gating and specialized training objectives. Closely related to our method, K-Merge (Shenaj et al., 2025) also manages a pool of LoRAs using threshold-based decisions to reuse or merge modules. However, K-Merge operates on already-trained LoRA adapters by calculating similarity in parameter weight-space, while CaLLM computes similarity on incoming data distributions in the embedding-space during live training, utilizing a least-used policy to intentionally overwrite dormant adapters instead of merging them.

**Prototypical networks and metric-based routing.** CaLLM's router is grounded in metric-based meta-learning (Vettoruzzo et al., 2024), drawing inspiration from Prototypical Networks. By computing distances to centroids in a learned embedding space, this non-parametric bias enables robust adaptation without inner-loop optimization. While prototypes have been used for class-incremental learning (De Lange & Tuytelaars, 2021; Zhu et al., 2021), CaLLM transposes this to task-level routing: each prototype represents the distributional signature of a specific adapter. This yields a training-free router that inherits the modularity of MoE but avoids the overhead of learned gating. By combining parameter isolation with an online, no-replay constraint, CaLLM provides a scalable alternative that handles novel tasks via prototype expansion and bounds memory through a repurposing policy.

## B. Extended Method Details

This appendix provides additional details on the CaLLM architecture that complement the overview in §2.

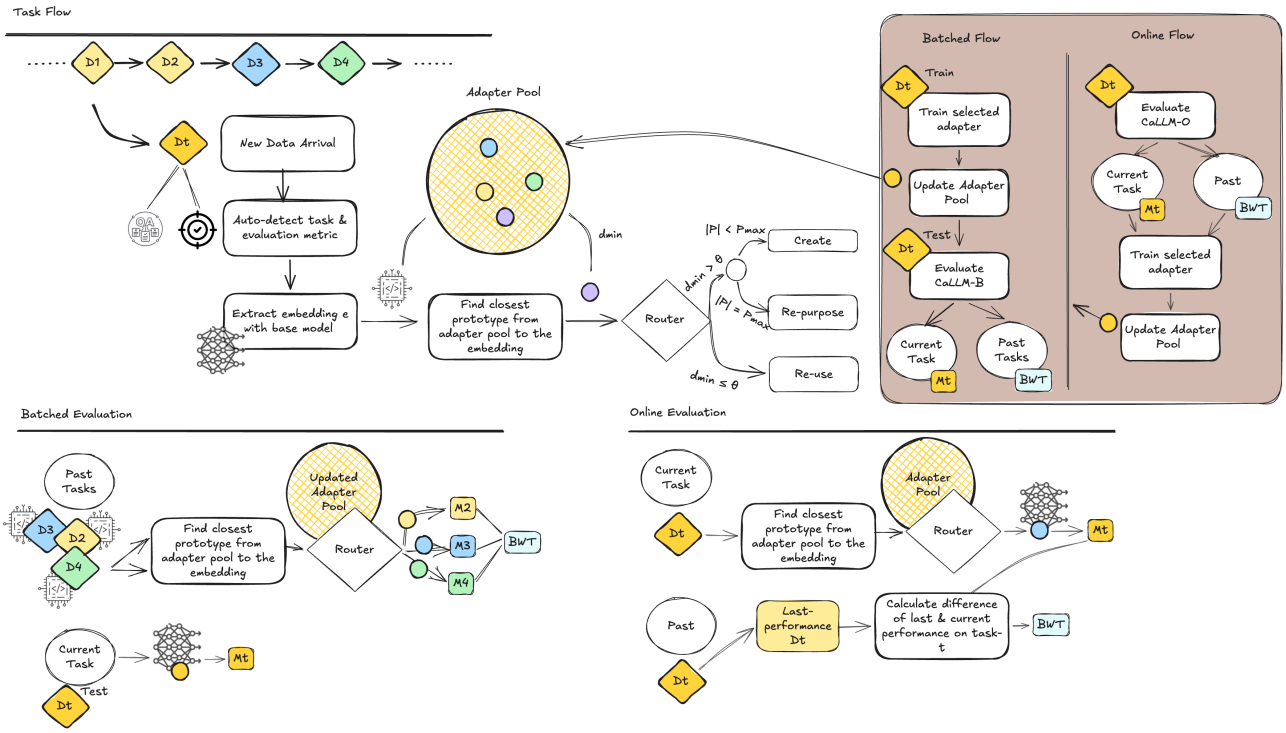



*Figure 3.* **CaLLM system overview (training + evaluation).** Each new data batch is embedded with a frozen base model (using mean-pooling). The non-parametric router manages a dynamic pool of specialized adapters, each mapped to a prototype. Based on semantic similarity between the new embedding and existing prototypes, the router triggers the reuse, creation, or repurposing of an adapter. The selected adapter is then trained or finetuned and the pool is updated. The diagram further details how this routing mechanism integrates with the distinct training and evaluation loops.

### B.1. Adapter Pool: Forward Pass

With LoRA adapters of rank $r$ and scaling $\alpha = 2r$, each linear projection $W$ in a Transformer block is augmented as

$$\mathbf{h}' = W\mathbf{x} + \frac{\alpha}{r} B_i A_i \mathbf{x}, \tag{4}$$

where $A_i \in \mathbb{R}^{d_{\text{in}} \times r}$ and $B_i \in \mathbb{R}^{r \times d_{\text{out}}}$ are the low-rank factors of the currently selected adapter $\ell_i$. All backbone parameters are frozen and only the active adapter's parameters are updated, ensuring that (i) the base model's representations remain stable, and (ii) adapters from different tasks do not interfere at the parameter level. Because only one adapter is active per forward pass, training FLOPs and GPU memory are independent of $|\mathcal{P}|$.

### B.2. Training Loss

Each step applies SFT on the selected adapter under the standard causal-LM objective:

$$\mathcal{L} = -\sum_{t=1}^{|\mathbf{y}|} \log P_{\theta_{\text{adapt}}}(y_t \mid y_{<t}, \mathbf{x}), \tag{5}$$

where $\mathbf{x}$ is the input prompt, $\mathbf{y} = (y_1, \ldots, y_{|\mathbf{y}|})$ is the target sequence, and $\theta_{\text{adapt}}$ are the adapter parameters. Optimizer, learning-rate schedule, gradient clipping, precision, and prompt formatting are decoupled from the routing logic; concrete settings are presented in Appendix D.2.

### B.3. Batched Task-Incremental Protocol

In Batched mode, tasks arrive sequentially as complete datasets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T$, each with train, evaluation, and test splits (Algorithm 1). For each task with dataset $\mathcal{D}_t$, the system first extracts an embedding from a sample of the training split and queries the router (with prototype updates enabled) to select or create an adapter. The selected adapter is then loaded onto the base model and fine-tuned on the full training split using the SFT objective (Eq. 5), with the evaluation split used for early stopping and checkpoint selection. After training, the adapter is saved to disk.

Evaluation proceeds in two phases. First, the trained adapter is evaluated on the current task's test set to measure adaptation quality. Second, to assess backward transfer, all previously seen tasks $j < t$ are re-evaluated: for each prior task, the router operates in read-only mode (no prototype updates) to find the best-matching adapter under the current prototype set, and that adapter is loaded and evaluated on task $j$'s test set. Backward transfer is then computed as the average normalized performance change across prior tasks.

### B.4. Online Continual Learning Protocol

In Online mode, data arrives as a stream of small chunks randomly interleaved across multiple tasks in a non-stationary distribution. (Algorithm 2). Each chunk contains a fixed number of examples ($chunk_size$) from a single task.

Following the *test-then-train* paradigm, the protocol begins with the router extracting a chunk's embedding $e_i$ to select, create, or repurpose an adapter according to the logic described above. Once the parameter subspace is determined, the chunk is immediately evaluated using the selected adapter, or the frozen backbone if a new adapter is being initialized, prior to any gradient updates. This pre-update score provides an unbiased measure of the system's current knowledge and is used to update the continual learning metrics defined below. After evaluation, the selected adapter is fine-tuned on the input chunk via SFT (Eq. 5). This single-pass procedure ensures structural consistency and strictly avoids post-update evaluation to prevent information leakage, allowing CaLLM to satisfy the core assumptions of online learning while tracking knowledge accumulation in real-time.

### B.5. Continual Learning Metrics

We report the following measures to quantify plasticity and stability across the task stream:

- **Average Performance:** In the batched setting, after learning task $t$, we compute the normalized average over all tasks seen so far (each evaluated with its best-matching adapter via read-only routing):

$$\text{Avg}_t = \tfrac{1}{t} \sum_{j=1}^{t} R_{j,t}/s_j,$$

  where $R_{j,t}$ is the performance on task $j$ after learning task $t$ and $s_j$ is the metric scale. This captures overall knowledge retention and adaptation quality at each point in the task sequence.

  In the online setting, the equivalent is the **Online Moving Average (OMA)**, a running normalized average updated at each step with the latest pre-update evaluation:

$$\text{OMA}_t = \tfrac{1}{t} \sum_{k=1}^{t} R_k^{\text{pre}}/s_k,$$

  where $s_k$ is the task-specific metric scale at step $k$. As a moving average over the stream, OMA tracks how the system's overall performance evolves in real-time, reflecting its ability to generalize to new distributions.

- **Backward Transfer (BWT):** Measures the impact of learning new tasks on previously learned ones. In the batched setting, after learning task $t$:

$$\text{BWT}_t = \tfrac{1}{t-1} \sum_{j=1}^{t-1} (R_{j,t} - R_{j,j})/s_j,$$

  where $R_{j,j}$ is the performance immediately after learning task $j$ and $s_j$ is the metric scale.

In the online setting, BWT is computed as:

$$\mathrm{BWT}_{j,t} = (R_j^{\text{pre},t} - R_j^{\text{pre,prev}})/s_j,$$

where $R_j^{\text{pre,prev}}$ is the pre-update score from the most recent prior observation of task $j$. Unlike the batched variant above, online BWT measures the change between consecutive observations of the same task rather than from the initial post-training score.

# C. Algorithms and Efficiency

## C.1. Algorithms (pseudocode)

**Algorithm 1** CaLLM-Batched

**Input:** task sequence $\mathcal{D}_1, \ldots, \mathcal{D}_T$; threshold $\eta$ (fixed or adaptive, Eq. 3); pool capacity $P$
**Output:** performance table $\mathbf{M}$
Initialize pool $\mathcal{P} = \varnothing$
**for** $t = 1$ **to** $T$ **do**
  *// Route (training mode: prototype updates enabled)*
  $\mathbf{e} \leftarrow \textsc{Embed}(\text{sample}(\mathcal{D}_t^{\text{train}}))$
  $i^*,\ d^* \leftarrow \arg\min_{i:\ell_i \in \mathcal{P}} d_i$  (Eq. 2; $d^* = \infty$ if $\mathcal{P} = \varnothing$)
  **if** $d^* \le \eta$ **then**
    $a_t \leftarrow \ell_{i^*};\ \mathbf{p}_{i^*} \leftarrow \frac{1}{2}(\mathbf{p}_{i^*} + \mathbf{e});\ c_{i^*} \leftarrow c_{i^*} + 1$  *// reuse*
  **else if** $|\mathcal{P}| < P$ **then**
    $a_t \leftarrow \textsc{NewAdapter}();\ \mathcal{P} \leftarrow \mathcal{P} \cup \{(a_t,\ \mathbf{e},\ 1)\}$  *// create*
  **else**
    $i_{\min} \leftarrow \arg\min_i c_i;\ a_t \leftarrow \ell_{i_{\min}};\ \mathbf{p}_{i_{\min}} \leftarrow \mathbf{e};\ c_{i_{\min}} \leftarrow 1$  *// repurpose*
  **end if**
  *// Train (overwrites adapter weights in the repurposing case)*
  $\text{SFT}(\mathcal{D}_t^{\text{train}},\ \mathcal{D}_t^{\text{eval}},\ \text{adapter}{=}a_t)$
  *// Evaluate current task*
  $\textsc{LoadAdapter}(a_t);\ \mathbf{M}[t,t] \leftarrow \textsc{Evaluate}(\mathcal{D}_t^{\text{test}})$
  *// Backward transfer on all previous tasks (read-only routing)*
  **if** $t > 1$ **then**
    **for** $j = 1$ **to** $t - 1$ **do**
      $\mathbf{e}_j \leftarrow \textsc{Embed}(\text{sample}(\mathcal{D}_j^{\text{test}}))$
      $a_j \leftarrow \ell_{\arg\min_i d_i(\mathbf{e}_j)}$  *// read-only: no prototype updates*
      $\textsc{LoadAdapter}(a_j);\ \mathbf{M}[j,t] \leftarrow \textsc{Evaluate}(\mathcal{D}_j^{\text{test}})$
    **end for**
    $\mathbf{M}[\text{BWT}, t] \leftarrow \frac{1}{t-1} \sum_{j=1}^{t-1} \big(\mathbf{M}[j,t] - \mathbf{M}[j,j]\big) \,/\, s_j$
  **else**
    $\mathbf{M}[\text{BWT}, t] \leftarrow 0$
  **end if**
  $\mathbf{M}[\text{Avg}, t] \leftarrow \frac{1}{t} \sum_{i=1}^{t} \mathbf{M}[i,t] \,/\, s_i$
**end for**

**Algorithm 2** CaLLM-Online

**Input:** chunk stream $(\tau_k, \mathcal{D}_k)$ for $k = 1, \ldots, N$; threshold $\eta$ (fixed or adaptive); pool capacity $P$
**Output:** performance table $\mathbf{M}$ with per-step BWT, and OMA
Initialize pool $\mathcal{P} = \varnothing,\ \text{last} = \{\},\ S_{\text{run}} = 0,\ c = 0$
**for** $k = 1$ **to** $N$, receiving $(\tau, \mathcal{D})$ **do**
  *// Route (single pass: prototype updates enabled)*
  $\mathbf{e} \leftarrow \textsc{Embed}(\text{sample}(\mathcal{D}))$
  $i^*,\ d^* \leftarrow \arg\min_{i:\ell_i \in \mathcal{P}} d_i \quad (d^* = \infty \text{ if } \mathcal{P} = \varnothing)$
  **if** $d^* \leq \eta$ **then**
    $a \leftarrow \ell_{i^*};\ \mathbf{p}_{i^*} \leftarrow \frac{1}{2}(\mathbf{p}_{i^*} + \mathbf{e});\ c_{i^*} \leftarrow c_{i^*} + 1$ *// reuse*
  **else if** $|\mathcal{P}| < P$ **then**
    $a \leftarrow \textsc{NewAdapter}();\ \mathcal{P} \leftarrow \mathcal{P} \cup \{(a,\ \mathbf{e},\ 1)\}$ *// create*
  **else**
    $i_{\min} \leftarrow \arg\min_i c_i;\ a \leftarrow \ell_{i_{\min}};\ \mathbf{p}_{i_{\min}} \leftarrow \mathbf{e};\ c_{i_{\min}} \leftarrow 1$ *// repurpose*
  **end if**
  *// Pre-update evaluation (test-then-train)*
  $\textsc{LoadAdapter}(a);\ \text{pre} \leftarrow \textsc{Evaluate}(\mathcal{D});\ \mathbf{M}[\tau(\text{pre}), k] \leftarrow \text{pre}$
  *// BWT (defined only when task has a prior observation)*
  **if** $\tau \in \text{last}$ **then**
    $\mathbf{M}[\text{BWT}, k] \leftarrow (\text{pre} - \text{last}[\tau]) \,/\, s_\tau$
  **end if**
  $\text{last}[\tau] \leftarrow \text{pre}$
  $S_{\text{run}} \leftarrow S_{\text{run}} + \text{pre} \,/\, s_\tau;\ c \leftarrow c + 1;\ \mathbf{M}[\text{OMA}, k] \leftarrow S_{\text{run}} \,/\, c$
  *// Train on same adapter, same chunk*
  $\text{SFT}(\mathcal{D},\ \text{adapter}{=}a)$
**end for**

### C.2. Computational and Memory Efficiency

A practical CL system must not only preserve performance but also remain efficient as the expert pool grows. Table 1 reports training FLOPs, throughput, and peak GPU memory for CaLLM at varying pool sizes alongside baselines, measured on the full TRACE-8 batched pipeline (gemma-3-12b-it, single NVIDIA RTX A6000 48 GB GPU, CUDA 12.4).

*Table 1.* **Compute and Memory Overhead on TRACE-8 (Batched).** CaLLM consumes the same training FLOPs as single-adapter Seq-FT regardless of pool size, with constant GPU memory. ER doubles wall-clock time due to replay, and oLoRA requires $2.1\times$ more GPU memory for orthogonal projections. †EWC, ER, and oLoRA use custom training loops that do not report `total_flos`; EWC's FLOPs are comparable to CaLLM (same data/rank), ER's are higher due to replay augmentation, and oLoRA's are lower due to smaller rank ($r$=4).

| Method | Pool $P$ | Train TFLOPs | Samples/s | Peak GPU (GB) | Wall (h) | # Adapters |
|---|---|---|---|---|---|---|
| CaLLM-B | 2 | 7,339 | 3.81 | 12.15 | 60.0 | 2 |
| CaLLM-B | 4 | 7,339 | 3.81 | 12.15 | 59.3 | 4 |
| CaLLM-B | 6 | 7,339 | 3.67 | 12.15 | 59.7 | 6 |
| CaLLM-B | 8 | 7,339 | 3.87 | 12.15 | 59.5 | 8 |
| Seq-FT | 1 | 7,339 | 3.53 | 12.15 | 68.2 | 1 |
| EWC† | 1 | — | — | 12.48 | 76.9 | 1 |
| ER† | 1 | — | — | 12.10 | 125.8 | 1 |
| oLoRA† | 8 | — | — | 25.20 | 75.1 | 8 |

**Training compute.** The central result is that CaLLM's training cost is *independent of pool size*: all four configurations ($P = 2, 4, 6, 8$) consume exactly 7,339 TFLOPs. This is a direct consequence of the modular architecture—only one adapter is active per forward pass, so the computational graph is identical regardless of how many adapters exist in the pool. Training throughput is also stable across pool sizes (3.67–3.87 samples/s), confirming that adapter switching introduces negligible latency.

CaLLM and Seq-FT share identical training FLOPs (7,339 TFLOPs), as expected—both perform the same number of gradient updates on the same data with the same adapter rank. The difference lies entirely in *where* gradients are routed, not in how many are computed. CaLLM achieves slightly higher throughput (3.81–3.87 vs. 3.53 samples/s) and lower wall-clock time (59–60 h vs. 68 h), likely because isolated adapters encounter simpler optimization landscapes than a shared adapter accumulating all tasks.

Among the baselines that explicitly address forgetting, each pays a distinct overhead. EWC trains on the same data with the same adapter rank, so its per-step FLOPs are expected to be comparable to CaLLM; the additional wall-clock cost (76.9 h) comes from Fisher information computation after each task. ER augments each training batch with replay buffer samples, which increases the effective data budget and thus is expected to incur higher total training FLOPs—this is reflected in its $2.1\times$ wall-clock overhead (125.8 h). oLoRA uses a smaller rank ($r$=4 vs. $r$=16), which should reduce per-step FLOPs, but must maintain orthogonal projection matrices across all subspaces simultaneously, making it the most memory-intensive baseline at 25.2 GB ($2.1\times$ CaLLM).

These results confirm that CaLLM achieves its anti-forgetting benefits at *zero additional training compute* relative to naïve sequential fine-tuning, while the baselines that attempt to mitigate forgetting through regularization, replay, or orthogonality constraints each pay a substantial cost in time, memory, or both.

**Routing overhead.** Unlike the baselines, CaLLM performs an additional forward pass through the frozen backbone for prototype matching (§ 2.2). Because this routing pass is encoder-only, i.e., no decoding, over a short input and reuses the already-loaded backbone, its marginal cost is small relative to the per-task training and inference compute. Using the standard transformer estimate of approximately $2N$ FLOPs per forward token, where $N \approx 12 \times 10^9$ for `gemma-3-12b-it`, and noting that each routing call processes $n \approx 20$ prompts of approximately 10 tokens each, one routing call costs $2 \cdot (12 \times 10^9) \cdot 20 \cdot 10 \approx 4.8 \times 10^{12}$ FLOPs, i.e., 4.8 TFLOPs. Over the full TRACE-8 batched run, with one routing call per task across 8 tasks, the router consumes $8 \cdot 4.8 \approx 38.4$ TFLOPs, or roughly $(38.4/7339) \times 100 \approx 0.5\%$ of the 7339 TFLOPs reported above for training. This is consistent with the observation that CaLLM and Seq-FT report identical training TFLOPs in Table 1: the router's contribution falls below the reporting precision. The same logic applies at inference time, where each routing call is amortized over all subsequent evaluation queries within a task, giving per-query overhead $\ll 0.1\%$, and during the online continual-learning phase, where routing is invoked once per chunk rather than per query. The routing module is, moreover, a tunable component: a smaller encoder or cached prompt embeddings could reduce this already-small overhead further, which we leave to future work.

**Parameter footprint.** All methods in Table 1 are highly parameter-efficient by construction: each trains well under 1% of gemma-3-12b-it's weights, and CaLLM, Seq-FT, EWC, and ER share a common LoRA $r = 16$ footprint per task (oLoRA's $r = 4$ is even smaller). The meaningful axis of comparison among CL methods is therefore not the raw trainable-parameter count but the auxiliary overhead each approach incurs from its forgetting-mitigation machinery (Fisher information for EWC, replay buffers for ER, and orthogonal projection matrices for oLoRA) which is precisely what the wall-clock and peak-memory columns capture.

**Memory management.** CaLLM maintains a constant 12.15 GB GPU memory footprint regardless of pool size, because exactly one adapter is loaded on GPU at any time. Inactive adapters are offloaded to CPU memory and swapped in on demand, so the GPU overhead does not grow with the number of adapters in the pool. The maximum pool capacity $P_{\max}$ can be set explicitly or computed automatically based on available system memory according to Eq. 6:

$$P = \left\lfloor \frac{M_{\text{avail}}}{S_{\text{adapter}}} \right\rfloor - 1, \tag{6}$$

where $M_{\text{avail}}$ is the available system (CPU) memory and $S_{\text{adapter}}$ is the on-disk size of a single adapter (both in GB); the $-1$ reserves one slot for the adapter currently resident on GPU. In the online regime, peak GPU memory remains flat at 12.01 GB—virtually identical to the batched setting—confirming that this constant-memory property holds across both training protocols.

**Online regime.** The online streaming configuration uses lighter adapters ($r$=4, $\alpha$=8) and single-epoch chunk updates, so its total FLOPs are not directly comparable to the batched runs above. A notable characteristic of the online setting is that *evaluation dominates the runtime*: out of 19.3 h total wall-clock time over 500 chunks, only 3.6 h (18.4%) is spent on

training while 13.9 h (72.3%) is consumed by the per-chunk test-then-train evaluation loop. This suggests that evaluation scheduling—rather than training efficiency—is the primary lever for reducing online deployment cost, and motivates future work on sparse or amortized evaluation strategies.

# D. Implementation Details

## D.1. Backbone selection

Table 2 compares zero-shot performance across three candidate backbones – Llama-3.1-8B-Instruct, gemma-3-12b-it, and Qwen3-8B– on the TRACE-8 benchmark. While all models struggle on the NumGLUE (math tasks), confirming the necessity of domain-specific adaptation, gemma-3-12b-it consistently exhibits the strongest generalist prior. It achieves the highest zero-shot accuracy across all eight tasks, with marked advantages on C-STANCE ($0.62$ vs. $0.20$), FOMC ($0.64$ vs. $0.14$), and Py150 ($14.91$ vs. $6.42$). Selecting gemma-3-12b-it as our primary backbone ensures that our results evaluate the efficacy of the CaLLM framework rather than compensating for a weak base-model initialization.

*Table 2.* **Inference-only (Zero-Shot) Performance on TRACE-8.** gemma-3-12b-it achieves the highest score on all 8 tasks. All models show low performance on math, motivating the need for continual adaptation.

| Task | Metric | Llama-3.1-8B-Instruct | gemma-3-12b-it | Qwen3-8B |
|---|---|---|---|---|
| C-STANCE | acc (0–1) | 0.20 | **0.62** | 0.20 |
| FOMC | acc (0–1) | 0.14 | **0.64** | 0.04 |
| MeetingBank | ROUGE-L (0–1) | 0.05 | **0.21** | 0.04 |
| Py150 | EDIM (0–100) | 6.42 | **14.91** | 0.00 |
| ScienceQA | acc (0–1) | 0.09 | **0.35** | 0.07 |
| NumGLUE-cm | acc (0–1) | 0.01 | **0.05** | 0.00 |
| NumGLUE-ds | acc (0–1) | 0.02 | **0.07** | 0.00 |
| 20Minuten | SARI (0–100) | 36.2 | **39.22** | 37.0 |

## D.2. Implementation details and hyperparameters

As detailed in Appendix D.1, all experiments use gemma-3-12b-it (4-bit quantized) as the primary backbone. We maintain this fixed model for all comparisons, though CaLLM is backbone-agnostic and compatible with any Transformer-based LLM. The router utilizes the same frozen backbone as a feature extractor, where the last hidden states are mean-pooled over valid tokens to obtain a sample embedding and then averaged across the input batch to produce a single representation for routing.

LoRA adapters are trained using the HuggingFace SFTTrainer with an 8-bit AdamW optimizer, gradient clipping at norm $1.0$, and bfloat16 mixed precision without sequence packing. Additional training constants include a maximum sequence length of $1024$, temperature of $1.0$, LoRA dropout of $0.05$, and a warmup ratio of $0.05$. In the online setting, we use $max_chunks = 500$ and $chunk_size = 40$, with chunks generated via random resampling. All inputs follow the Alpaca prompt style detailed in §3.1, and the router uses the adaptive threshold defined in Eq. 3.

Batched experiments are reported as the mean and standard deviation across three random seeds. Online experiments are instead performed as a single run due to the significant computational cost of $500$-chunk streams. We rely on the OMA metric, which averages over $500$ evaluation steps, to provide inherent statistical stability. Table 3 summarizes the specific hyperparameters selected via grid search over learning rate $\in \{10^{-4}, 5 \times 10^{-4}, 10^{-3}\}$, LoRA rank $\in \{4, 8, 16\}$, and epochs $\in \{3, 5\}$. All experiments were conducted on a single NVIDIA RTX A6000 GPU with 48GB of VRAM using `Unsloth` (Han et al., 2023) and the `trl`, `peft` and `transformers` libraries from HuggingFace.

*Table 3.* **CaLLM hyperparameters per scenario.** All runs use the HuggingFace SFTTrainer with 8-bit AdamW, gradient clipping at norm 1.0, and bfloat16 mixed precision. LR = learning rate, $r$ / $\alpha$ = LoRA rank / scaling factor, Batch = batch size, GAS = gradient accumulation steps, LR Sched. = learning rate schedule.

| Setting | LR | $r$ | $\alpha$ | Epochs | Batch | GAS | LR Sched. |
|---|---|---|---|---|---|---|---|
| CaLLM-B, TRACE | $1\times10^{-4}$ | 16 | 32 | 3 | 16 | 4 | cosine |
| CaLLM-B, CITB | $1\times10^{-3}$ | 8 | 16 | 5 | 25 | 4 | cosine |
| CaLLM-O, TRACE | $3\times10^{-3}$ | 4 | 8 | 1 | 10 | 2 | linear |
| CaLLM-O, CITB | $1\times10^{-3}$ | 8 | 8 | 1 | 20 | 1 | cosine |

### D.3. Details about the baselines

To evaluate CaLLM, we compare it against several representative strategies from the CL literature, adapted for the PEFT regime. To ensure a fair and rigorous comparison, all baselines share the same core architectural parameters as CaLLM and the same LoRA configuration (rank, alpha, and dropout). We then utilize the optimal hyperparameters (such as regularization strength or buffer size) specific to each method as reported in their respective original papers, unless stated otherwise.

- **Inference only** is a simple baseline where the base model (e.g. gemma-3-12b-it) is used during inference in a continual/online way, every time a new task/chunk is presented to the model.
- **Sequential Adapter Fine-Tuning (Seq-FT)**: A single LoRA adapter is fine-tuned sequentially on the task stream. This represents the lower bound for stability, as it is highly susceptible to catastrophic forgetting.
- **Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017)**: A regularization-based approach that penalizes changes to parameters that were important for previous tasks. Importance is estimated via the diagonal of the Fisher Information Matrix. We apply EWC to the LoRA weights specifically.
- **Experience Replay (ER) (Rolnick et al., 2019):** A memory buffer stores a subset of samples from previous tasks. During the adaptation to a new task, each training batch is augmented with replayed samples from the buffer to preserve past knowledge. In our experiments, we use a fixed buffer size of $1000$ samples.
- **Orthogonal LoRA (OLoRA) (Wang et al., 2023a):** A subspace isolation method that constrains the LoRA weight updates to be orthogonal to the pre-trained weights or previous task directions, aiming to prevent gradient-based interference. To ensure compatibility with the Gemma backbone and avoid out-of-memory (OOM) errors, we modified the original OLoRA implementation as follows: (i) reduced the evaluation batch size to $4$, instead of using the batch size, and (ii) offloaded the orthogonal loss calculation to the CPU. For the TRACE-8 online stream, we utilized $batch_size = 8$, $r = 4$, $\alpha = 8$. For CITB, we used $batch_size = 2$, $r = 4$, $\alpha = 8$, and gradient accumulation step of $8$.

### D.4. gemma-3-12b-it training data and CITB overlap

Gemma 3 models are well-suited for text-generation tasks such as question answering, summarization, and reasoning. The 12B model used in our work was trained on 12T tokens consisting of web documents (in over 140 languages), code, mathematics, and images. The pre-training data is not released openly and there is limited information about the data sources. Additionally, the IT variant has been instruction-tuned using knowledge distillation from larger teacher models (likely Gemini models).

CITB is built on top of the Super-NaturalInstructions dataset (Wang et al., 2022). In this work, we focus on the InstrDialog task stream, which consists of 19 dialogue-related tasks spanning three categories: dialogue state tracking (4 tasks), dialogue generation (11 tasks), and intent identification (4 tasks). All tasks are evaluated using the ROUGE-L metric.

While there is no clear information about the data used by Gemma 3 for pre-training or fine-tuning, it is plausible that the Super-NaturalInstructions dataset (on which CITB is based) was included in the training data of either Gemma 3 or the Gemini teacher models. This would explain why LoRA fine-tuning on CITB does not improve over the inference-only baseline. Despite this, all other baselines degrade under sequential fine-tuning, suggesting that CaLLM maintains the original performance even when the data has already been seen.

To investigate potential data contamination, we applied the method of Shi et al. (2023), which computes per-token probabilities under the hypothesis that "seen" examples have fewer low-probability outliers with respect to "unseen" words. We compared 500 CITB samples against 309 "old" Wikipedia pages (likely within Gemma 3's training data) and 97 Wikipedia pages created in the following 60 days (unlikely to be in the training data). The results are as follows:

- Member mean: 0.005722
- Non-member mean: 0.003475
- CITB mean: 0.002848

From these results we can conclude that CITB has an average probability closer to the non-member samples than the (assumed) member samples, suggesting that CITB is likely not in the training data of Gemma 3.

This appears to contradict our earlier hypothesis that Gemma 3 has already seen CITB data. Indeed, Figure 4b shows that fine-tuning with CITB data does not improve over the inference-only baseline. However, it is important to note that the method of Shi et al. (2023) is only an empirical heuristic, and the candidate member and non-member sets are based on assumptions about Wikipedia page creation dates rather than ground-truth training data membership.

### D.5. Synthetic data generation for CITB

The original CITB benchmark, defined as InstrDialog in the original paper (Zhang et al., 2023), consists of 19 tasks derived from Super-NaturalInstructions, detailed in Table 4.

*Table 4.* The 19 dialogue tasks in the CITB InstrDialog stream.

| Category | Task Type | Count |
|---|---|---|
| Dialogue State Tracking | Slot Filling, State Update | 4 |
| Dialogue Generation | Response Gen, Clarification | 11 |
| Intent Identification | Classification, Intent Detection | 4 |
| **Total** | | **19** |

Each task contains only 500 training instances, which is insufficient for effectively training LoRA adapters and creates an imbalance when compared to the TRACE-8 benchmark (5000 instances per task). To ensure a fair comparison and improve adaptation stability, we generated synthetic training data for each CITB task. Inspired by self-synthesized rehearsal strategies (Huang et al., 2024), we utilized the frozen gemma-3-12b-it model as a data generator. We employed a few-shot prompting strategy, providing the model with two randomly sampled real instances in-context and tasking it to generate a novel example following the same Alpaca-style format. By iterating this process with randomized in-context examples, we produced an initial pool of 6000 synthetic instances per task.To ensure data quality and distributional consistency, we applied a two-stage refinement pipeline:

1. **Deduplication:** We removed all exact string matches and near-duplicates to ensure diversity within the synthetic pool.
2. **Centroid-based filtering:** To retain only high-quality, representative samples, we converted the synthetic instances into embeddings using the frozen base model. We then applied K-means clustering to the embeddings of each task. Finally, we calculated the Euclidean distance between each instance and its respective cluster centroid, selecting the 3000 instances closest to the centroid for the final training set.

This procedure resulted in a refined CITB dataset containing 3000 examples per task. This refined version maintains the semantic integrity of the original tasks while providing sufficient signal for parameter-efficient fine-tuning, and it serves as the standard CITB data stream used for all experiments and baseline comparisons presented in this paper.

### D.6. Automatic Metric Selection

When ground-truth metric labels are unavailable (i.e., in label-free stream configurations), CaLLM dispatches an automatic metric selection procedure at the start of each task's evaluation phase. This adds only a negligible time overhead from a single short inference call and no memory overhead, as it reuses the already-loaded backbone. The procedure works as follows.

**Model.** The same base LLM used for task inference (gemma-3-12b-it, 4-bit quantized) serves as the zero-shot classifier for metric selection. This avoids the overhead of loading an auxiliary model and the system simply executes a single generation call before the model training and evaluation on each task.

**Meta-prompt template.** A single representative prompt is sampled from the task's test split, truncated to 700 characters to manage context, and embedded into a structured meta-prompt. The prompt instructs the LLM to categorize the task based on its linguistic and structural features (e.g., presence of code-specific keywords, multiple-choice indicators, or summarization requests). The model is constrained to a closed set of responses: `ACCURACY`, `ROUGE`, `SARI`, or `EDIM`.

```
Analyze the following task prompt and select the best evaluation metric.
RULES:
1. Code completion tasks (contains 'code', 'python', 'complete the code', etc.)
→ Use EDIM
2. Summarization tasks (contains 'summarize', 'summary', 'write a summary', etc.)
→ Use ROUGE
3. Multiple choice questions (contains 'A)', 'B)', 'C)', 'choose', etc.) → Use
ACCURACY
4. Text simplification tasks (contains 'simplify', 'simplified version', etc.) →
Use SARI
5. Default for all other tasks → Use ACCURACY
Task prompt: {prompt}
Respond with ONLY the metric name. Choose one: ACCURACY, ROUGE, SARI, or EDIM
### Response:
```

**Parsing and fallback.** To ensure robustness, the model's output is parsed for the first occurrence of a valid metric keyword. If the model fails to produce a recognized metric name within its response, the system defaults to `ACCURACY` as a conservative baseline. Once a metric is selected, it is cached for the duration of the task's residency in the stream to prevent redundant computation.

# E. Additional Experimental Results

## E.1. Batched Task-Incremental Performance

This appendix expands the batched results summarised in §3.2.

**Per-task TRACE-8 results.** Table 5 reports the final per-task scores after sequential training over the full 8-task TRACE benchmark. CaLLM-B achieves the highest score among all baseline methods on 6 of 8 tasks. The most significant gains over the inference-only baseline appear on specialized domains where the pre-trained backbone is weakest: code completion (Py150, $+2.33$ in EDIM), science reasoning (ScienceQA, $0.35 \rightarrow 0.81$ in accuracy), and math (NumGLUE-ds, $6\times$ improvement in accuracy). Conversely, on C-STANCE and 20Minuten, performance is largely inherited from the backbone; on the former, fine-tuning slightly degrades pre-existing competence, while on the latter all methods fall within a narrow 2-point SARI band. Because 20Minuten is the final task in the sequence, no method suffers from forgetting on it, narrowing the gap between CaLLM-B and baselines.

*Table 5.* **Per-Task Final Performance on TRACE-8 (Batched).** Final scores after training on all tasks sequentially.

| Task | Metric | Inf-Only | Seq-FT | EWC | ER | oLoRA | CaLLM-B |
|---|---|---|---|---|---|---|---|
| C-STANCE | acc (0–1) | **0.62** | 0.03 | 0.28 | 0.04 | 0.09 | 0.43 |
| FOMC | acc (0–1) | 0.64 | 0.30 | 0.31 | 0.14 | 0.06 | **0.68** |
| MeetingBank | ROUGE-L (0–1) | 0.21 | 0.06 | 0.10 | 0.10 | 0.16 | **0.23** |
| Py150 | EDIM (0–100) | 14.91 | 11.96 | 10.55 | 12.98 | 9.81 | **17.24** |
| ScienceQA | acc (0–1) | 0.35 | 0.13 | 0.61 | 0.19 | 0.00 | **0.81** |
| NumGLUE-cm | acc (0–1) | 0.05 | 0.14 | 0.12 | 0.06 | 0.04 | **0.15** |
| NumGLUE-ds | acc (0–1) | 0.07 | 0.28 | 0.15 | 0.13 | 0.05 | **0.42** |
| 20Minuten | SARI (0–100) | 39.22 | 38.06 | **40.06** | 39.63 | 39.91 | 39.49 |

**Per-task CITB results.** Table 6 disaggregates the final CITB-19 results by individual task, analogous to Table 5 for TRACE-8. All 19 tasks use ROUGE-L (0–1) as the evaluation metric. Scores are averaged across 3 seeds for all methods except EWC (2 seeds available). CaLLM-B achieves the highest score among all continual learning methods on 12 of 19 tasks. On the remaining 6, oLoRA leads on individual tasks but trails CaLLM-B in mean performance (0.24 vs. 0.31), indicating that its per-task wins come at the cost of broader forgetting elsewhere. The high Inference-Only scores on tasks such as Diplomacy (T1590), P.Chat Gen. (T1729), and Mutual Dial. (T611) reflect strong backbone priors that sequential fine-tuning erodes for all methods.

**Average normalized performance over the task sequence.** Figure 4 shows the evolution of the average normalized score as tasks are added sequentially. On TRACE-8, baselines struggle with radical distribution shifts: EWC degrades sharply when encountering tasks that require more complex reasoning (*Py150* at Task-4), while ER and oLoRA exhibit a steady downward trend in later tasks. In contrast, CaLLM-B remains stable throughout the sequence by routing incompatible distributions (e.g., code vs. math) to separate experts. CaLLM-B finishes the 8-task sequence with an average normalized score of 0.44, substantially outperforming the Inference-Only baseline (0.31) and ER (0.31), with narrow $\pm 1$ SD variance bands. On CITB-19 (Fig. 4b), the challenge shifts to sequence length. While Seq-FT degrades rapidly due to interference within a single adapter, CaLLM-B maintains a stable trajectory across all 19 tasks: by instantiating new experts as needed ($P \approx 19$), the system prevents parameter overloading. CaLLM-B achieves the highest score among CL methods on 12 of 19 tasks, demonstrating broad competence across fine-grained CITB dialogue subtypes (see Table 6). However, fine-tuning on CITB-19 does not improve over Inference-Only for any method, likely because dialogue tasks lie within the natural competence of a 12B instruction-tuned model (see Appendix D.4).

*Table 6.* **Per-Task Final Performance on CITB-19 (Batched).** Scores measured after all 19 tasks have been trained sequentially; all tasks use ROUGE-L (0–1) averaged over 3 seeds.

| Task | Inf-Only | Seq-FT | EWC | ER | oLoRA | CaLLM-B |
|---|---|---|---|---|---|---|
| Diplomacy (T1590) | **0.84** | 0.14 | 0.05 | 0.20 | 0.10 | 0.31 |
| ConvAI3-1713 | **0.36** | 0.05 | 0.07 | 0.14 | 0.18 | 0.28 |
| CIRCA (T565) | 0.08 | 0.10 | 0.04 | 0.09 | 0.28 | **0.40** |
| P.Chat Choose (T1730) | 0.02 | 0.24 | 0.02 | 0.09 | 0.39 | **0.45** |
| AirDial. Class. (T573) | **0.92** | 0.05 | 0.07 | 0.08 | 0.27 | 0.36 |
| Story Motiv. (T294) | 0.34 | 0.09 | 0.07 | 0.09 | **0.49** | 0.25 |
| DSTC3 Class. (T1500) | 0.12 | 0.05 | 0.00 | 0.03 | 0.12 | **0.44** |
| SMCalFlow-1603 | **0.68** | 0.12 | 0.06 | 0.12 | 0.11 | 0.47 |
| Curiosity Dial. (T576) | 0.25 | 0.28 | 0.03 | 0.04 | **0.29** | 0.28 |
| MultiWOZ (T639) | 0.14 | 0.07 | 0.10 | 0.08 | **0.32** | 0.15 |
| ConvAI3-1714 | 0.20 | 0.11 | 0.02 | 0.23 | 0.12 | **0.25** |
| DSTC3 Ans. (T1501) | 0.06 | 0.05 | 0.03 | 0.07 | 0.31 | **0.41** |
| P.Chat Gen. (T1729) | **0.82** | 0.16 | 0.08 | 0.07 | 0.18 | 0.09 |
| AirDial. Gen. (T574) | **0.22** | 0.11 | 0.05 | 0.09 | 0.16 | 0.20 |
| PubMedQA (T848) | 0.08 | 0.17 | 0.04 | 0.05 | 0.06 | **0.21** |
| Mutual Dial. (T611) | **0.84** | 0.05 | 0.12 | 0.19 | 0.36 | 0.26 |
| Craigslist (T766) | 0.21 | 0.13 | 0.05 | 0.12 | 0.11 | **0.36** |
| DealOrNo (T1384) | 0.07 | 0.12 | 0.10 | 0.10 | 0.26 | **0.38** |
| SMCalFlow-1600 | 0.26 | 0.36 | 0.19 | 0.28 | **0.50** | 0.35 |

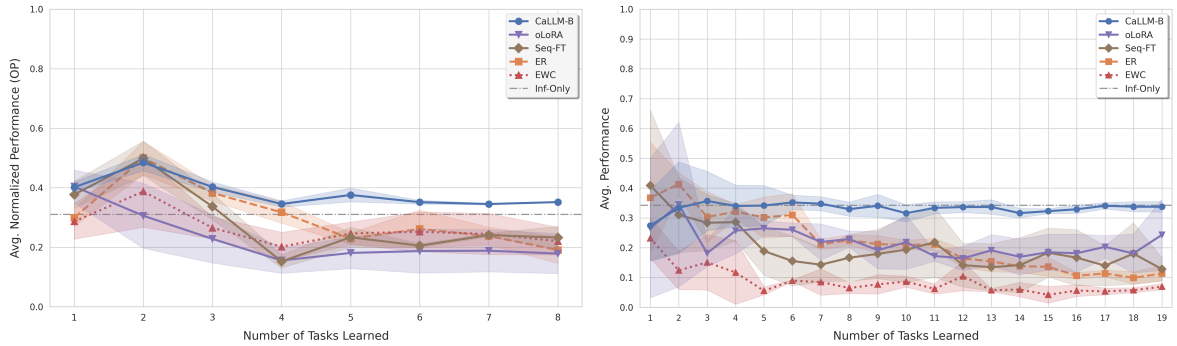



*Figure 4.* **Task-Incremental Learning Performance (mean ±1 std).** Average normalized score as tasks are added sequentially, for CaLLM-B and all baselines. **(a)** TRACE-8. **(b)** CITB-19.

### E.2. Online Streaming Heatmap

Figure 5 shows the complete 500-chunk task×expert assignment heatmap for the TRACE online stream. Each row corresponds to a task, each column to a chunk, and colour encodes the expert ID assigned by the router. The "null" entries (dark blue) indicate chunks in which the corresponding task did not appear in the interleaved stream. Despite the sparsity, the per-task colour patterns are stable throughout the full stream, confirming the persistent task—expert binding reported in §3.2. The zoomed view (chunks 200–300) used in the main text (Figure 1(b)) was selected as a representative window that minimises null-cell density and maximises readability.

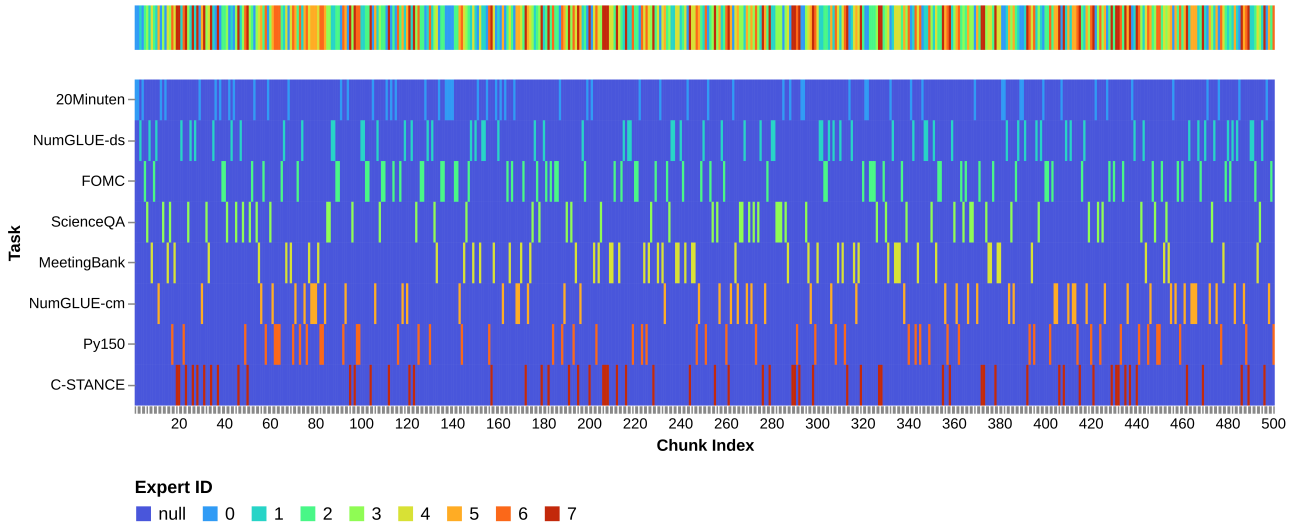



*Figure 5.* **Full Expert-Assignment Heatmap (TRACE, 500 chunks, P=8).** Each row is a task; each column a chunk; colour encodes the assigned expert ID. Dark blue ("null") cells indicate chunks where that task was absent from the stream. The per-task colour patterns remain stable across the full 500-chunk stream, confirming persistent task–expert binding.

### E.3. Why modular routing works

CaLLM's ability to mitigate interference in task-agnostic settings depends on the router's capacity to recognize distributional boundaries from input features alone. Figure 6 visualizes 2D t-SNE projections of the frozen backbone's mean-pooled features for both benchmarks. On TRACE-8 (left), the router produces well-separated clusters, with radical domain shifts (e.g. code vs. multilingual text) pushed far apart in embedding space. This geometric separation is expected given the large distributional gaps between TRACE tasks, but it confirms that the router's lightweight, training-free architecture combined with a base-model representation is sufficient to capture these boundaries without any additional finetuning. On CITB-19 (right), the router discriminates between 19 fine-grained dialogue sub-tasks (dialogue state tracking, generation, and intent identification) despite their shared conversational format, demonstrating that the intrinsic geometric separation also holds at finer granularity. This intrinsic separation is what enables reliable, task-agnostic routing: when pool capacity $P$ is sufficient, each cluster maps to a dedicated adapter, achieving full parameter isolation. The consequences of restricting $P$ and forcing cluster merges are analyzed in §3.3 and visualized in Appendix E.4.

Figure 6 illustrates the inherent separability of the task distributions within the router's embedding space. Using t-SNE projections of the frozen backbone's mean-pooled features, we observe that diverse tasks (both for TRACE-8 and CITB-19) form well-defined, distant clusters.

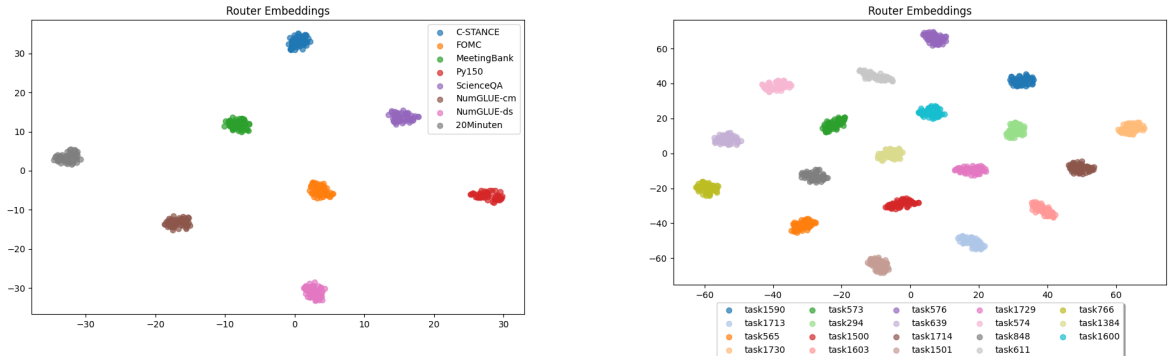
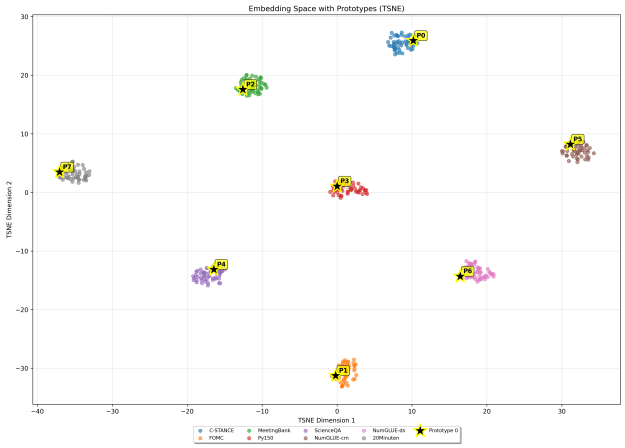



*Figure 6.* **Router Embedding Space (2D Projection).** t-SNE projections of the router's embeddings, coloured by task identity. **(Left)** TRACE-8: 8 tasks spanning diverse domains. **(Right)** CITB-19: 19 dialogue sub-tasks. Both benchmarks exhibit well-separated clusters, confirming that the frozen-backbone embedding alone supplies the distributional signal needed for task-agnostic adapter selection.

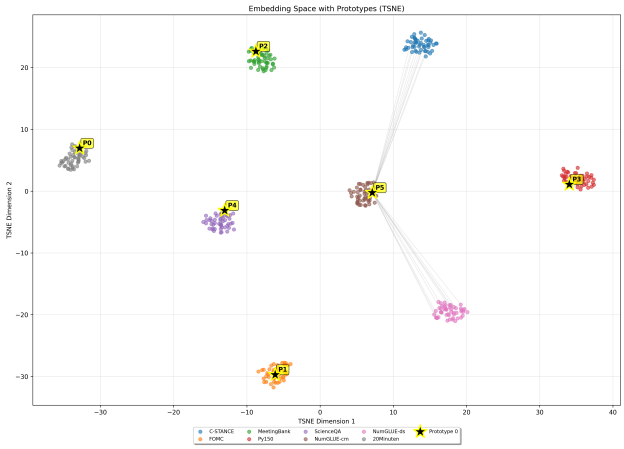
### E.4. Prototype Dynamics and Capacity Progression

Figure 7 demonstrates how the router dynamically partitions the embedding space under varying capacity constraints, studied in §3.3, for TRACE-8 . Each panel overlays $P$ prototypes (yellow stars) on the task-cluster structure; grey lines connect each data point to its nearest prototype. The prototype positions determine which tasks share an expert during training.

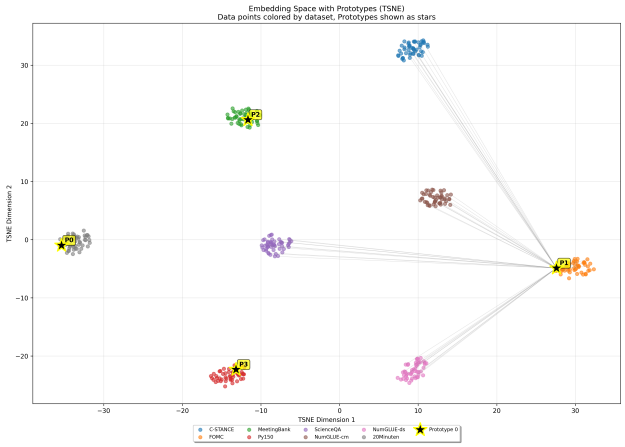
At $P{=}8$ (Figure 7a), every prototype settles at the centroid of one task cluster, giving zero-interference isolation. At $P{=}6$ (Figure 7b), the two NumGLUE math tasks and the C-STANCE dataset are pulled toward a shared prototype, a constructive merge that incurs negligible BWT. At $P{=}4$ (Figure 7c), one prototype absorbs five semantically diverse tasks on the right side of the space, producing the conflicting gradients that drive destructive overload. Even here, Py150 (code) retains its own isolated prototype (in red), confirming the router's sensitivity to the most extreme distributional gap. At $P{=}2$ (Figure 7d), only 20Minuten keeps a dedicated prototype; all other tasks collapse onto a single expert.



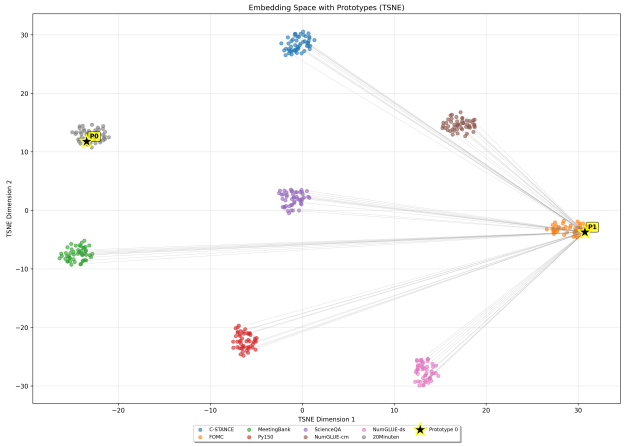

*(a)* $P$=8: perfect isolation, one prototype per task cluster.




*(b)* $P$=6: NumGLUE-cm/ds share a prototype—constructive merge, negligible BWT.



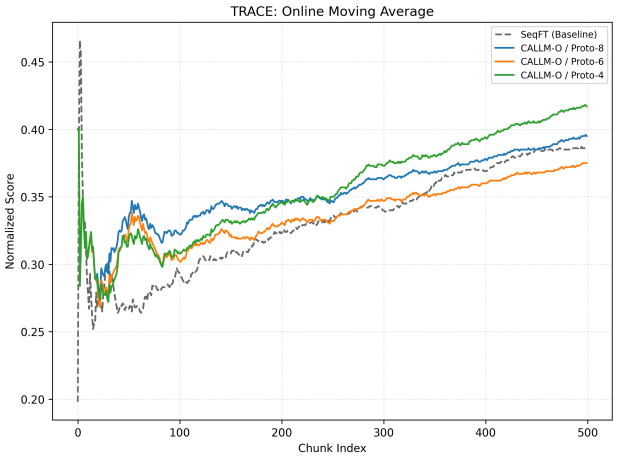

*(c)* $P$=4: one prototype absorbs 5 diverse tasks—destructive overload, BWT $= -0.19$.



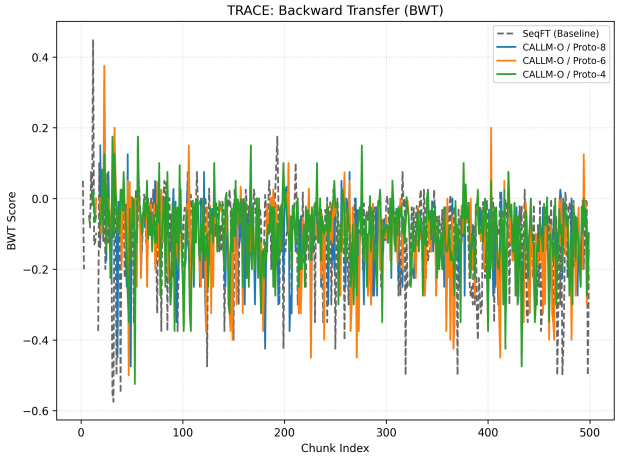

*(d)* $P$=2: system collapse, 7 of 8 tasks share one prototype, BWT $= -0.25$.

*Figure 7.* **Router Embedding Space with Prototypes (TRACE-8, t-SNE).** Yellow stars are prototype positions; grey lines show nearest-prototype assignment. The transition from $P$=8 to $P$=2 maps directly onto the isolation → constructive clustering → destructive overload → collapse progression described in §3.3.

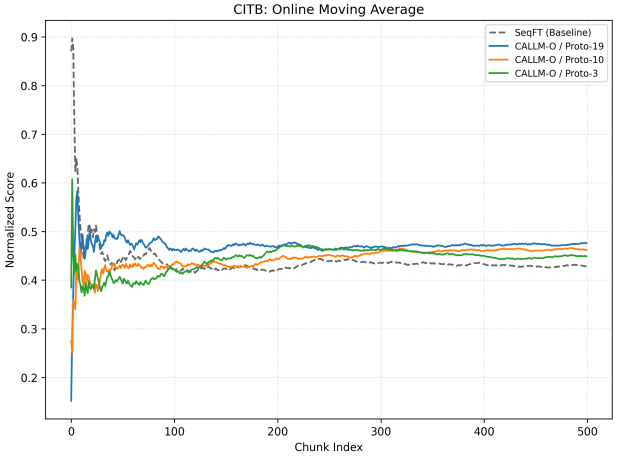
### E.5. Online Pool Constraints: OMA and BWT Curves



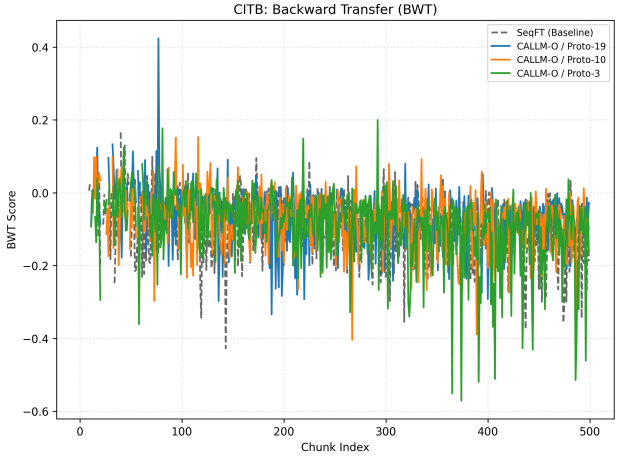

*(a)* TRACE OMA ($P \in \{4, 6, 8\}$)




*(b)* TRACE BWT ($P \in \{4, 6, 8\}$)




*(c)* CITB OMA ($P \in \{3, 10, 19\}$)




*(d)* CITB BWT ($P \in \{3, 10, 19\}$)

*Figure 8.* **Online Pool Constraints (500-chunk stream).** OMA and BWT for CaLLM-O and the SeqFT baseline. **Top row (TRACE-8):** (a) OMA and (b) BWT for pool sizes $P \in \{4, 6, 8\}$. **Bottom row (CITB-19):** (c) OMA and (d) BWT for pool sizes $P \in \{3, 10, 19\}$. Knowledge accumulation (OMA) is successful across all configurations; performance is non-monotonic with respect to pool size on TRACE, while CITB shows comparable cumulative scores but diverging forgetting profiles.

### E.5.1. Aggregate Online Routing Matrices

Figure 9 shows confusion matrices for TRACE ($P \in \{4, 6, 8\}$) with the fraction of chunks routed to each expert over a 500-chunk stream. As analyzed in §3.3, these matrices reveal that at $P{=}4$, the router settles into highly stable, semantically aligned groups that outperform the $P{=}6$ configuration due to reduced task displacement.

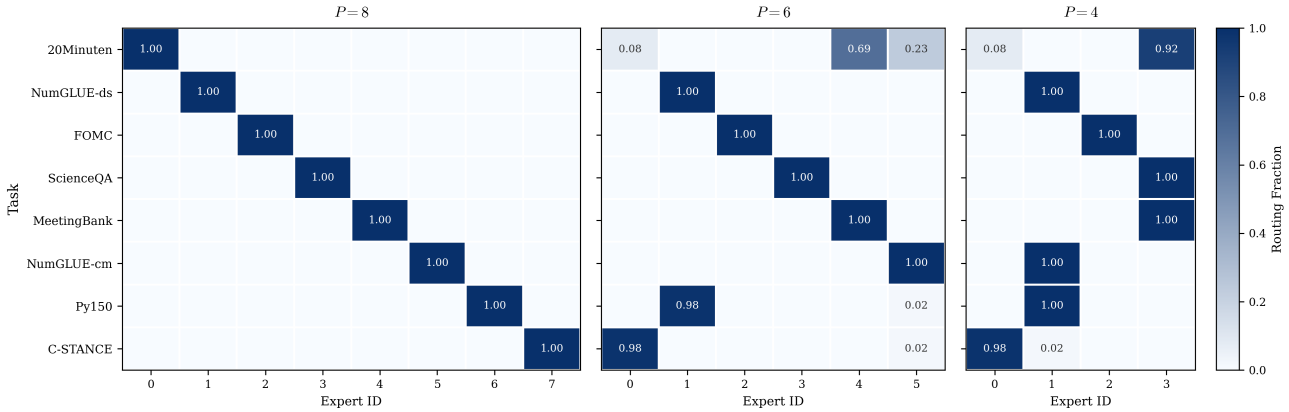



*Figure 9.* **Router Assignment Matrices on TRACE (Online).** Pool sizes $P \in \{4, 6, 8\}$. Each cell shows the fraction of chunks from that task routed to each expert.

## F. Limitations and Future Directions

While CaLLM demonstrates effective continual adaptation under the conditions evaluated in this work, some open questions remain and motivate future research.

**Backbone selection.** Our experiments use gemma-3-12b-it as the frozen foundational backbone for all CaLLM variants and baselines, selected based on the zero-shot comparison in Appendix D.1. This choice is deliberate and central to our framing: CaLLM is designed not to utilize a single ever-larger model to handle every new task, but to enable continual adaptation around a modestly-sized frozen backbone via a constant-overhead learning regime built on task-specialized adapters. Our motivation is rooted in efficiency and sustainability concerns as well as growing evidence that smaller models, when properly specialized, can match much larger generalists on individual tasks. That said, the non-parametric prototype router relies on mean-pooled final-layer hidden states from this backbone (Eq. 1), and the geometry of that embedding space directly governs routing dynamics: prototype distances, adaptive thresholding (Eq. 3), and the discriminability of incoming task distributions all depend on backbone representational quality. Different model families (e.g., Llama, Qwen, Mixtral) and scale regimes (sub-1B to 70B+) may exhibit distinct embedding geometries which could alter both the optimal pool size and threshold sensitivity. A systematic sweep across backbones and scales is therefore a natural follow-up, and we view characterizing how routing dynamics shift across this axis as a productive next step rather than a constraint on the framework.

**Adapting hyperparameters.** Eq. 6 provides a memory-driven upper bound on $P$, and our experiments reveal a non-monotonic dependence of performance on pool size (e.g., $P{=}4$ outperforms $P{=}6$ on TRACE). We examined this regime in detail and identified the underlying mechanism. What we do not yet offer is automating the discovery of this optimum from input data. A promising next step is therefore to integrate online AutoML to adapt currently fixed hyperparameters (including pool size, distance threshold, and learning rate) as the stream unfolds, paired with explicit merging of related adapters to consolidate shared knowledge. Together, these would let the effective pool respond to underlying task structure rather than be fixed in advance.

**Benchmark coverage.** Our two benchmarks are deliberately complementary along orthogonal axes: TRACE-8 stresses high task variety with relatively few tasks, while CITB-19 stresses a longer stream of more homogeneous, low-discrepancy tasks. CaLLM performs well in both regimes, indicating that the framework handles task diversity and stream length independently. A valuable addition to the continual learning benchmarks in this domain is to evaluate on streams that combine both axes simultaneously, very long streams interleaving high- and low-diversity tasks.

**Modality scope.** We focus on text-only streams (TRACE-8 and CITB-19), since language is the natural starting point when working with LLMs. Foundation models are nevertheless increasingly multimodal, and continual adaptation in vision-language or audio-language settings introduces additional challenges such as cross-modal drift, where updating one encoder can degrade reasoning grounded in another. Extending prototype-based routing to multi-modal streams spanning vision, language, and other modalities simultaneously, and characterizing forward and backward transfer when multiple modalities co-exist in the pool, is therefore a promising next step.

**Adversarial robustness.** Because CaLLM's repurposing strategy relies on a deterministic least-used eviction policy, it is theoretically susceptible to adversarial thrashing. An adversary could construct a specific sequence of alternating task prompts designed to deliberately overwrite a targeted adapter, only to immediately present data requiring that recently deleted module. This would force the system into a costly loop of continuous weight overwriting and recreation, degrading long-term performance. While defending against intentional adversarial manipulation is outside the scope of this work, developing robust eviction policies, such as randomized replacement, threshold-based pinning of critical modules, or introducing an external buffer, presents a crucial and necessary direction for deploying modular continual learning systems in security-sensitive environments.