Local Mixtures of Experts: Essentially Free Test-Time Training via Model Merging

Ryo Bertolissi[*], **Jonas Hübotter**[*], **Ido Hakimi, Andreas Krause**
ETH Zürich, Switzerland

Abstract

Mixture of expert (MoE) models are a promising approach to increasing model capacity without increasing inference cost, and are core components of many state-of-the-art language models. However, current MoE models typically use only few experts due to prohibitive training and inference cost. We propose ***T**est-**T**ime **M**odel **M**erging* (TTMM) which scales the MoE paradigm to an order of magnitude more experts and uses model merging to avoid almost any test-time overhead. We show that TTMM is an approximation of test-time training (TTT), which fine-tunes an expert model for each prediction task, i.e., prompt. TTT has recently been shown to significantly improve language models, but is computationally expensive. We find that performance of TTMM improves with more experts and approaches the performance of TTT. Moreover, we find that with a 1B parameter base model, *TTMM is more than* $100\times$ *faster than TTT* at test-time by amortizing the cost of TTT at train-time. Thus, TTMM offers a promising cost-effective approach to scale test-time training.

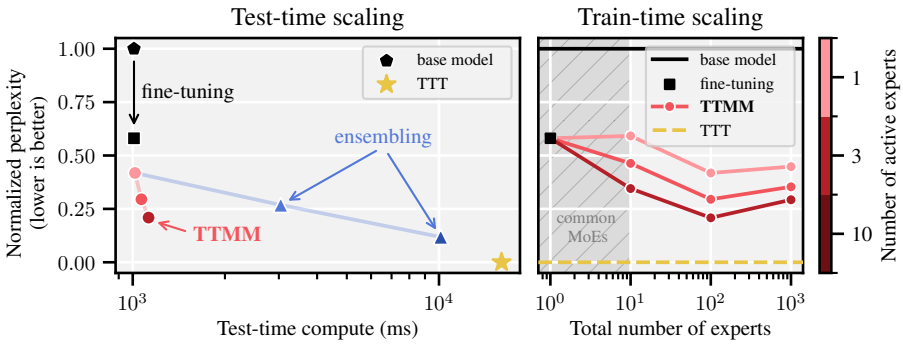

Figure 1: Accuracy gains of TTMM when scaling test-time compute (left) and pre-training compute (right). The normalized perplexity between base model and TTT model is averaged across evaluation datasets and models. **Left:** We measure language modeling ability against the time to generate 100 tokens. TTMM approaches the performance of TTT without almost any test-time overhead when increasing the number of active experts. Ensembling rather than merging experts improves performance, but at a much higher test-time cost. Standard fine-tuning on the training data improves performance without any test-time overhead, but does not yield close to the performance of TTT. **Right:** We measure the performance of TTMM against the total number of experts. Whereas common MoE models use few experts, TTMM scales to an order of magnitude more experts, improving performance with more experts. Scaling the number of experts in TTMM does not affect training FLOPs or required GPU memory.[1] Training the individual experts is embarrassingly parallelizable.

[*]Equal contribution. Correspondence to jonas.huebotter@inf.ethz.ch.
[1]Scaling the number of experts increases CPU host memory, which is cheaper than GPU memory.

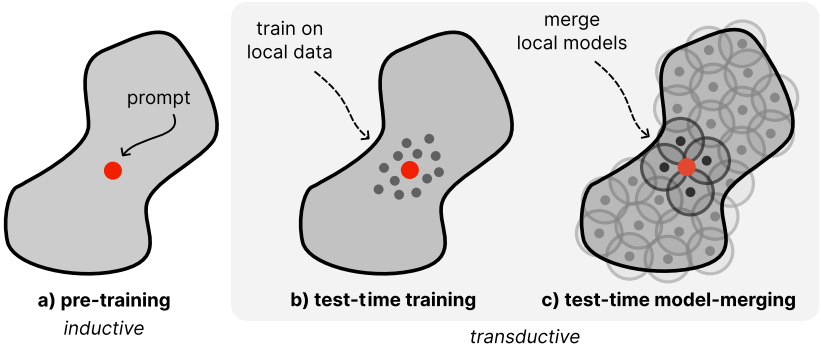

Figure 2: Standard *inductive* language modeling pre-trains a single model on the dataset (shown in gray) which is then used for prediction. Test-Time Training (TTT) improves performance by fine-tuning the model at test-time on data related to the prompt, but is computationally expensive. Test-Time Model Merging (TTMM) fine-tunes the pre-trained model to many local neighborhoods at train-time, and merges the local models related to the prompt at test-time. Both TTT and TTMM target models specifically to the prompt (i.e., are *transductive*), yet, TTMM does not require expensive fine-tuning at test-time.

1 Introduction

The standard paradigm of machine learning separates training and testing. In training, we learn a model by *inductively* extracting general rules from data, while in testing, we apply this model to new unseen data. We study an alternative *transductive* paradigm called *test-time training* (TTT, Sun et al., 2020) where we use a specific expert model for each prediction task. Variations of this paradigm have been studied since the inception of the machine learning field (Cleveland, 1979; Cleveland & Devlin, 1988; Atkeson et al., 1997; Bottou & Vapnik, 1992). More recently, fine-tuning large pre-trained neural networks at test-time has regained interest in computer vision (e.g., Jain & Learned-Miller, 2011; Shocher et al., 2018; Sun et al., 2020; Gandelsman et al., 2021; Ruiz et al., 2023) and language modeling (Krause et al., 2018; 2019; Hardt & Sun, 2024; Sun et al., 2024; Hübotter et al., 2025). Hardt & Sun (2024) and Hübotter et al. (2025) fine-tune a pre-trained language model on data related to the prompt, and show that this can significantly improve language modeling performance. Beyond pure language modeling, TTT has been instrumental in state-of-the-art approaches to abstract reasoning (Akyürek et al., 2025), video generation (Dalal et al., 2025), and also been shown to improve reasoning language models (Zuo et al., 2025; Simonds & Yoshiyama, 2025).

While TTT can significantly improve model performance, it comes with a high computational cost at test-time: TTT requires fine-tuning the model for every task, i.e., each prompt. We propose ***T**est-**T**ime **M**odel **M**erging* (TTMM) which amortizes the cost of TTT at train time:

- At **train-time**, TTMM clusters the training data into many local neighborhoods and trains separate small expert LoRA adapters (Hu et al., 2022) for each cluster.
- At **test-time**, TTMM dynamically selects a subset of LoRA adapters and merges their parameters to form a single task-specific model.

In our experiments, TTMM approaches the language modeling performance of TTT without incurring almost any significant compute or memory cost (cf. Figure 1). Figure 2 illustrates the difference between TTT and TTMM.

Another extensive body of work studies merging multiple models to combine distinct capabilities and knowledge (Yang et al., 2024b). We discuss the connection of TTMM to this literature on model merging and mixture of experts more extensively in Section 2. Model merging is typically used in a multitask setting with *few* "meta-tasks" (e.g., coding, math, law, etc.) and corresponding expert models. This multitask setting differs from the TTT setting where each individual prompt is considered its own task. Hence, TTMM trains *many* local models, even within a single meta-task (say for all kinds of different coding tasks). Then, for *each* prompt at test-time, TTMM merges the most related local models. As such, TTMM can

be seen as bridging multitask model merging and TTT by dynamically constructing an expert model at test-time from an order of magnitude more meta-tasks than in multitask learning.

We contribute the following key findings, summarized in Figure 1:

1. **TTMM outperforms multitask model merging:** We find that training and merging *many* expert models, even within a single domain such as "coding in Python", outperforms training and merging *few* models specialized on broader tasks.
2. **TTMM approaches performance of TTT without almost any test-time overhead:** We discuss how to merge the parameters of the expert models at almost no additional compute or memory cost. We then evaluate TTMM on the Wikipedia and GitHub Python corpora and find that it achieves performance close to TTT, while matching the performance of ensembling without incurring its test-time overhead.

2 Related Work

We propose TTMM as an efficient amortization of TTT, which realizes comparable accuracy. We next discuss how TTMM can be seen as an extension to the literature on model selection and model merging. In Appendix A, we discuss the link to retrieval-augmented generation.

Multitask Model Merging. Combining the predictions of multiple models has been a long-standing practice in machine learning, e.g., in the form of majority voting (Wang et al., 2023) or by ensembling of predictions (Dietterich, 2000; Wortsman et al., 2022), which has been used to avoid catastrophic forgetting (e.g., Bagatella et al., 2025). However, merging predictions of many neural networks is computationally inefficient since it requires a separate forward pass for each model. Instead, TTMM merges in parameter-space, and only requires a single forward pass of the merged model, resulting in a negligible increase of compute and memory cost compared to inferencing a single model. The idea of merging multiple models by averaging their parameters (not their predictions!) is grounded in the surprising phenomenon of *mode-connectivity* (Garipov et al., 2018; Wilson, 2025), which observes that modes in ensembles of models are connected by paths of small loss. This has motivated a variety of model merging techniques, including stochastic weight averaging (Izmailov et al., 2018) and model soups (Wortsman et al., 2022). Recently, model merging has been studied predominantly in the context of merging *few* expert models with the aim of retaining performance on their respective meta-tasks (Ilharco et al., 2023; Matena & Raffel, 2022; Jin et al., 2023; Ainsworth et al., 2023). Merging model parameters may lead to interference, which has been a key focus of recent work (Yadav et al., 2023; Yang et al., 2024c; Tang et al., 2024b; Daheim et al., 2024; Yu et al., 2024; Jung et al., 2024).

Dynamic Merging of Multitask Models. Recent work in multitask model merging studies selecting merging coefficients dynamically depending on the prompt. Oh et al. (2025) select coefficients based on the entropy of the individual model's predictions, which is computationally expensive since it requires forward passes of all networks. Instead, TTMM computes merging coefficients based on a measure of similarity between prompt and experts which is tuned on holdout data at train-time (Lu et al., 2024; Tang et al., 2024a; Cheng et al., 2024; Lai et al., 2025), and which incurs only negligible test-time overhead. In the tangentially related literature on black-box model fusion, dynamic model selection has been widely studied (e.g., Liu et al., 2021; Jiang et al., 2023; Mavromatis et al., 2024). However, in the setting of TTMM, we control data and architecture used for training the local models, which enables us to compute merging coefficients more efficiently than in black-box model fusion.

Clustering and Dynamic Model Selection. While the above literature focuses on multitask learning with typically only a handful of expert models, TTMM clusters the training data into *many* local neighborhoods and trains a separate local model for each cluster. Like TTMM, Branch-Train-Merge (BTM, Li et al., 2022; Gururangan et al., 2023) clusters the training data into local neighborhoods, trains a separate local model for each cluster, and dynamically selects a subset of these models at test-time. However, unlike BTM, TTMM uses an order of magnitude more local models and merges their parameters instead of their predictions, leading to substantially faster inference. In concurrent work, Qiu et al. (2025) study TTMM in a continual learning setting.

Figure 3: Illustration of **TTMM**: At *train-time* (Algorithm 1), TTMM clusters the training data (shown in gray) into many local neighborhoods and trains a separate expert model for each cluster. At *test-time* (Algorithm 2), TTMM dynamically selects a subset of expert models related to the prompt and merges their parameters to form a single task-specific model. The test-time stage is compute and memory efficient, using the previously trained experts.

Mixture of Experts. The mixture of experts (MoE, Shazeer et al., 2017) paradigm increases model capacity without increasing inference cost by introducing conditional routing of inputs to subsets of learnable parameters (i.e., "experts"). Training MoE models is challenging due to the need for joint training of the experts and the router on substantial multitask data. Moreover, MoE models evaluate all active experts individually, which can still require substantial memory. This in turn implies that the number of experts is typically limited to a small number. Recent studies have focused on challenges such as reducing synchronization (Sukhbaatar et al., 2024; Zhang et al., 2024b; 2025), load balancing of experts (Clark et al., 2022; Zhou et al., 2022), and improving expert specialization (Dai et al., 2024). In contrast, TTMM approximates the MoE paradigm by embarrassingly parallelizing the training of the local models and merging model parameters at test-time, which permits scaling to many orders of magnitude more experts and which enables approximation of the "maximally local" TTT.

3 TTMM: Test-Time Model Merging

Our proposed method, TTMM, has two main stages: train-time and test-time. At train-time, TTMM clusters the training data into many local neighborhoods and trains a separate expert model for each cluster. Then at test-time, TTMM dynamically composes these expert models to form a single task-specific model. We discuss both stages in more detail below.

We consider a domain $\mathcal{X}$ of token sequences and a training set $\mathcal{D} \subseteq \mathcal{X}$. We further assume that we have access to a pre-trained autoregressive language model $f(\boldsymbol{x}; \boldsymbol{\theta})$ that maps token sequences $\boldsymbol{x} \in \mathcal{X}$ to probability distributions over the next token. Finally, we assume access to a sequence embedding model $\boldsymbol{\phi}(\boldsymbol{x})$ returning normalized embeddings. In our experiments, we use a mean-pooled sequence embedding model, but one could also use the last-layer token embeddings from the pre-trained language model (see Lu et al., 2024).

Train-time: Clustering. To cluster the training dataset, we use bisecting k-means, a hierarchical clustering algorithm which recursively applies k-means with $k = 2$ to the cluster with the largest diameter (Jain, 2010). We use bisecting k-means since it is more efficient than k-means and tends to lead to more balanced clusters, however, any clustering algorithm can be used. We then train a small LoRA adapter on each cluster for one epoch. Finally, we average the embeddings of the sequences in each cluster

Algorithm 1 TTMM at train-time

Require: Language model $f(\boldsymbol{x}; \boldsymbol{\theta})$ with pre-trained weights $\boldsymbol{\theta}$; sequence embedding model $\boldsymbol{\phi}(\boldsymbol{x})$; training set $\mathcal{D}$; number of clusters K.

1: Cluster training set $\mathcal{D}$ into K clusters $\mathcal{D}_1, \ldots, \mathcal{D}_K$.
2: Train a LoRA adapter $\boldsymbol{\theta}_k$ on each cluster $\mathcal{D}_k$.
3: Compute cluster-specific embeddings:

$$\boldsymbol{\phi}_k \leftarrow \text{normalize}\Big(\tfrac{1}{|\mathcal{D}_k|} \textstyle\sum_{\boldsymbol{x} \in \mathcal{D}_k} \boldsymbol{\phi}(\boldsymbol{x})\Big)$$

to obtain cluster-specific embeddings (i.e., "centroids") that represent the capabilities of the corresponding expert model. We renormalize the centroids to avoid biasing the embedding norm of larger clusters toward zero. Without the additional normalization, centroids of larger clusters have smaller norm despite $\boldsymbol{\phi}(\boldsymbol{x})$ being normalized.

Test-time: Model Merging. At test-time, given a prompt $\boldsymbol{x}^\star \in \mathcal{X}$ and its embedding $\boldsymbol{\phi}^\star \doteq \boldsymbol{\phi}(\boldsymbol{x}^\star)$, we compute merging coefficients w_k for each expert model $\boldsymbol{\theta}_k$ using a sparse cross-attention mechanism with sparsity parameter $\tau \in [0, \frac{1}{K})$:

$$\text{sparse-softmax}_\tau(\boldsymbol{z}) \doteq \frac{\text{ReLU}(\text{softmax}(\boldsymbol{z}) - \tau)}{\sum_k \text{ReLU}(\text{softmax}(\boldsymbol{z})_k - \tau)}$$

where $\text{ReLU}(z) = \max\{0, z\}$. This sparse softmax returns a sparse distribution over the experts, where experts with weight below τ are pruned, and the remaining weights are renormalized. Pruning experts with low weights improves test-time efficiency since less expert parameters need to be merged (see Figure 4 (right)), with minimal degradation of accuracy. The sparsity being smaller than $1/K$ ensures that always at least one expert is selected. The model selection can easily be parallelized for all clusters and multiple queries by writing the computation as a cross-attention matrix product:

$$\boldsymbol{\Theta}^\star \leftarrow \text{sparse-softmax}_\tau\left(\frac{1}{\beta}\boldsymbol{\Phi}^\star\boldsymbol{\Phi}^\top\right)\boldsymbol{\Theta}$$

with keys $\boldsymbol{\Phi} \doteq [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_K]^\top \in \mathbb{R}^{K\times d}$, queries $\boldsymbol{\Phi}^\star \doteq [\boldsymbol{\phi}_1^\star, \ldots, \boldsymbol{\phi}_n^\star]^\top \in \mathbb{R}^{n\times d}$, and values $\boldsymbol{\Theta} \doteq [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K]^\top \in \mathbb{R}^{K\times p}$, computing the collection of n merged expert models $\boldsymbol{\Theta}^\star \in \mathbb{R}^{n\times p}$.

Algorithm 2 TTMM at test-time

Require: Prompt $\boldsymbol{x}^\star$ with embedding $\boldsymbol{\phi}^\star$. Language model $f(\boldsymbol{x};\boldsymbol{\theta})$ with pre-trained weights $\boldsymbol{\theta}$; expert models $\{(\boldsymbol{\theta}_k, \boldsymbol{\phi}_k)\}_{k=1}^K$; temperature β and sparsity τ (tuned on holdout data).

1: Compute cluster-specific merging coefficients:
$$w_k \leftarrow \text{sparse-softmax}_\tau\left(\tfrac{1}{\beta}\boldsymbol{\phi}_k^\top\boldsymbol{\phi}^\star\right)$$
2: Merge into a single expert model:
$$\boldsymbol{\theta}^\star \leftarrow \textstyle\sum_{k=1}^K w_k\boldsymbol{\theta}_k$$
3: Use $f(\boldsymbol{x}^\star;\boldsymbol{\theta}^\star)$ to generate the next token.

3.1 Latency

In contrast to MoE models, the number of experts in TTMM is typically too large for all experts to fit in GPU memory at once. For example, with Llama-3.2-1B as base model, each LoRA adapter has size approximately 172 MB, and with 1000 clusters, the expert models would require prohibitive 172 GB of GPU memory. Instead, we dynamically load active experts into GPU memory and merge their parameters before inference.

The test-time overhead of TTMM consists therefore of the time required to compute the merging coefficients (*select*), the time to load active experts from CPU to GPU memory (*load*), and the time to merge the expert parameters (*merge*). In Figure 4 (middle), we benchmark this overhead across varying numbers of active experts. On our infrastructure and with 10 active experts, selection takes roughly 5 milliseconds, loading takes around 30 milliseconds, and merging takes approximately 80 milliseconds. We use PyTorch's `einsum` (Paszke, 2019) for computing the merged LoRA adapter $\Delta\boldsymbol{W} = \sum_k w_k\boldsymbol{B}_k\boldsymbol{A}_k$ (cf. Figure 7 in the appendix), with $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ the low-rank factors of the LoRA adapter $\boldsymbol{\theta}_k$. This is around 20% faster than the naive implementation with a for-loop over the experts, without any additional memory overhead. We anticipate that test-time overhead may be reduced further by interleaving loading and merging by overlapping computation and communication (see, e.g., Tang et al., 2025).

TTMM costs ≈20 tokens. In Figure 4 (middle), we evaluate the practical test-time overhead of TTMM for language generation. TTMM incurs only a small fixed overhead which, when merging 10 experts, is equal to generating around 20 tokens. When generating more than 100 tokens, the overhead of TTMM compared to inference with the base model is negligible. In contrast, ensembling has a multiplicative overhead that grows with the number of active experts. Comparing to TTT, a single test-time gradient step with Llama-3.2-1B takes approximately 150 milliseconds (Hübotter et al., 2025). With 100 test-time gradient steps, the total test-time overhead (without any data selection and loading) is already around 15 seconds. In contrast, *TTMM with* 10 *active experts has a constant overhead of around* 115 *milliseconds, which is a more than* 125× *speedup compared to TTT*.

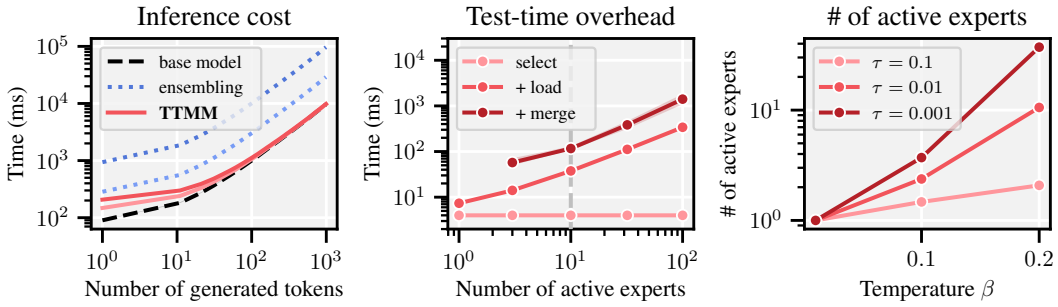

Figure 4: TTMM has negligible compute overhead at test-time. **Left:** Inference cost of ensembling and TTMM with 10 (dark) and 3 (light) active experts, respectively. TTMM with 10 experts costs around 20 generated tokens, while with 3 experts, TTMM costs only about 10 generated tokens. In contrast, ensembling has multiplicative overhead that grows with the number of active experts. **Middle:** Test-time overhead of TTMM compared to inference with the base model. When using up to 10 experts, the test-time overhead is small. **Right:** Number of active experts with a given temperature β and sparsity τ on the Wikipedia dataset with $K = 1000$ experts. **Setting:** We use Llama-3.2-1B as base model and consider LoRA adapters with rank 64. Each LoRA adapter has size $\approx$172 MB. For loading, we assume a CPU to GPU bandwidth of 50 GB/s. For computing embeddings, we use the `all-mpnet-base-v2` model. We measure selection and merging on an RTX 4090 GPU with 10 seeds.

Balancing Latency and Performance. The exact Pareto frontier of latency and performance varies depending on generation length. Figure 1 plots the Pareto frontier for 100 generated tokens and Figure 4 (left) shows how the latency changes for other generation lengths. We do not show TTT in Figure 4 (left) since the constant overhead of TTT before generating any token is larger than the time to generate 1k tokens with TTMM (and 10 active experts).

3.2 Test-Time Model Merging approximates Test-Time Training

TTMM can be seen as an approximation of TTT that amortizes TTT's compute cost at train-time. We formalize this connection by showing that in the limit of exponentially many experts, TTMM closely approximates TTT (Hardt & Sun, 2024), which fine-tunes the pre-trained model on the N nearest neighbors to the prompt in the training data. To see the intuition behind this approximation, first consider the set of all subsets of $\mathcal{D}$ of size N. Suppose we pre-train an expert model on each of these subsets. Then, at test-time, we select the expert model whose training data embeddings have the shortest distance to the prompt embedding. Note that this is *equivalent* to TTT, without the need for fine-tuning at test-time. We have simply pre-computed all possible expert models at train-time, and at test-time fetch the right one. TTMM approximates this interpretation of TTT in three steps.

Approximation 1: Fewer amortized experts. The number of pre-trained expert models in the above scenario is not practically feasible, as it grows exponentially with the training data. Thus, TTMM considers a more practical scenario with a smaller number of clusters where we do not pre-compute all possible TTT models. Our following proposition indicates that even an expert model different to the TTT model, yet trained on a cluster $\mathcal{D}'$ sufficiently close to the prompt embedding, may still be a good approximation of TTT:

Informal Proposition 3.1 (Formalized in Proposition C.1)**.** *Suppose the neural network $f(\boldsymbol{x};\boldsymbol{\theta})$ is L-Lipschitz in $\boldsymbol{\theta}$ and that the loss is G-smooth in $\boldsymbol{x}$, with any $L, G > 0$. Further, suppose we adapt models via $T \geq 1$ steps of gradient descent with step size $\eta > 0$ from a shared initialization $\boldsymbol{\theta}$. For any prompt $\boldsymbol{x}^\star \in \mathcal{X}$ and $N \geq 1$, let $\boldsymbol{\theta}_{\boldsymbol{x}^\star}$ be the TTT model trained on the set $\mathcal{D}_{\boldsymbol{x}^\star}$ of the N nearest neighbors to $\boldsymbol{\phi}(\boldsymbol{x}^\star)$ in $\mathcal{D}$. Let $\boldsymbol{\theta}'$ be the expert model trained on any $\mathcal{D}' \subseteq \mathcal{D}$. Then, if $\mathcal{D}'$ contains at least one nearest neighbor from $\mathcal{D}_{\boldsymbol{x}^\star}$,*

$$\left\| f(\boldsymbol{x}^\star;\boldsymbol{\theta}_{\boldsymbol{x}^\star}) - f(\boldsymbol{x}^\star;\boldsymbol{\theta}') \right\| \leq \eta TLG(\operatorname{diam}(\mathcal{D}_{\boldsymbol{x}^\star}) + \operatorname{diam}(\mathcal{D}')). \tag{1}$$

To understand the intuition behind this proposition, consider an expert model trained on a cluster $\mathcal{D}'$ that contains parts of the local neighborhood of the prompt embedding (for this,

the entire clustering must *cover* the dataset). Then, the above proposition shows that this expert model is a good approximation of the TTT model if the diameter of $\mathcal{D}'$ is also *small*. Consequently, this motivates the approximation of TTT by TTMM with many expert models *covering* the entire dataset, each trained on a *small* cluster. Training expert models on this partitioning of the dataset does not cost more FLOPs or GPU memory than training a single model on the entire dataset. This contrasts with TTT, which considers a separate expert model for each possible recombination of the training data into subsets of size N.

Approximation 2: Summarizing experts with their centroids. Instead of selecting the expert model by minimizing the sum of distances of individual data points to the prompt embedding (i.e., $\sum_{\boldsymbol{x}\in\mathcal{D}_k} \|\boldsymbol{\phi}(\boldsymbol{x}^\star) - \boldsymbol{\phi}(\boldsymbol{x})\|$), TTMM summarizes each expert model by its centroid $1/|\mathcal{D}_k| \sum_{\boldsymbol{x}\in\mathcal{D}_k} \boldsymbol{\phi}(\boldsymbol{x})$. Selecting the expert with the closest centroid to the prompt embedding further speeds up inference, since we only need to compute K distances, rather than the distances to all training data embeddings. This simplification does not come for free: Consider the one-dimensional set of embeddings $\{-1, 0, 1\}$ and a prompt embedding of 0 with $N = 2$. Then, the optimal cluster is either $\{-1, 0\}$ or $\{0, 1\}$, yet TTMM would pick $\{-1, 1\}$ since its centroid averages out differences in the embeddings. Since TTMM first clusters the data, there is reason to expect that each cluster has small diameter, in which case the centroid is a good approximation of the individual data points.

Approximation 3: Merging multiple experts. As a final step, to compensate for the approximation error of the chosen expert model induced by steps (1) and (2), TTMM merges multiple expert models. Intuitively, and as visualized in Figure 3, this is motivated by multiple experts being able to cover distinct features of the prompt that may not all be captured by a single expert model. We find empirically that merging a small number of experts can improve model performance without incurring a significant test-time overhead (cf. Figure 1). TTMM computes merging coefficients via cross-attention, which is equivalent to weighting the active experts with an RBF kernel:[2]

$$\text{softmax}\left(\frac{1}{\beta}\boldsymbol{\phi}_k^\top \boldsymbol{\phi}^\star\right) = \text{normalize}\left(\exp\left(-\frac{1}{2\beta}\|\boldsymbol{\phi}^\star - \boldsymbol{\phi}_k\|_2^2\right)\right).$$

Weighting data (not models!) by an RBF kernel has long been a common approach in machine learning and used, e.g., in kernel density estimation (Rosenblatt, 1956; Parzen, 1962), kernel regression (Nadaraya, 1964; Watson, 1964), RBF networks (Lowe & Broomhead, 1988; Moody & Darken, 1989), and local learning (e.g., Bottou & Vapnik, 1992; Atkeson et al., 1997). TTMM additionally sparsifies the selected models to minimize the test-time compute overhead due to loading and merging.

4 Results

We evaluate language modeling with causal language models. We report the perplexity of the models on holdout data from the Wikipedia (English, Wikimedia Foundation, 2025) and GitHub (Python code) corpora, comprising 6.4M and 7.2M documents, respectively. As base models, we use Llama-3.2-1B (Grattafiori et al., 2024) and Qwen2.5-1.5B (Yang et al., 2024a). For clustering and model selection, we use embeddings from the `all-mpnet-base-v2` model (Reimers & Gurevych, 2019; Song et al., 2020). Following Hardt & Sun (2024) and Hübotter et al. (2025), we evaluate TTT by training a LoRA adapter for a single gradient step each on the 100 nearest neighbors to the prompt embedding, most to least similar. We also compare against a single model, which is fine-tuned for a single epoch on the respective corpus. This fine-tuned model can be considered an upper bound on the performance of approaches to multitask learning, since multitask learning typically trains one model per meta-task (i.e., per corpus) and merging models from multiple meta-tasks typically reduces performance on any individual meta-task.

Our code is available at `https://github.com/rbertolissi/ttmerge`. We share fine-tuned baselines and TTMM-MoEs on Huggingface.

[2]See Appendix B for a proof.

Model	Wikipedia		GitHub (Python)	
	Merging	*Ensembling*	*Merging*	*Ensembling*
Llama-3.2-1B	8.674		2.611	
+ fine-tuning	7.849		2.581	
+ TTMM (1 active expert)	7.669		2.552	
+ TTMM (3 active experts)	7.571	7.579	2.519	2.512
+ **TTMM** (10 active experts)	**7.510**	7.509	**2.492**	2.464
+ TTT	7.559		2.441	
Qwen2.5-1.5B	8.570		2.335	
+ fine-tuning	7.702		2.317	
+ TTMM (1 active expert)	7.166		2.330	
+ TTMM (3 active experts)	7.101	7.096	2.304	2.294
+ **TTMM** (10 active experts)	**7.080**	7.061	**2.287**	2.257
+ TTT	7.231		2.177	

Table 1: Perplexity (lower is better) across evaluation datasets and base models with 100 experts. **Bold** numbers denote the best test-time efficient method, and underlined numbers denote the best overall. TTMM outperforms standard multitask fine-tuning of a single model.

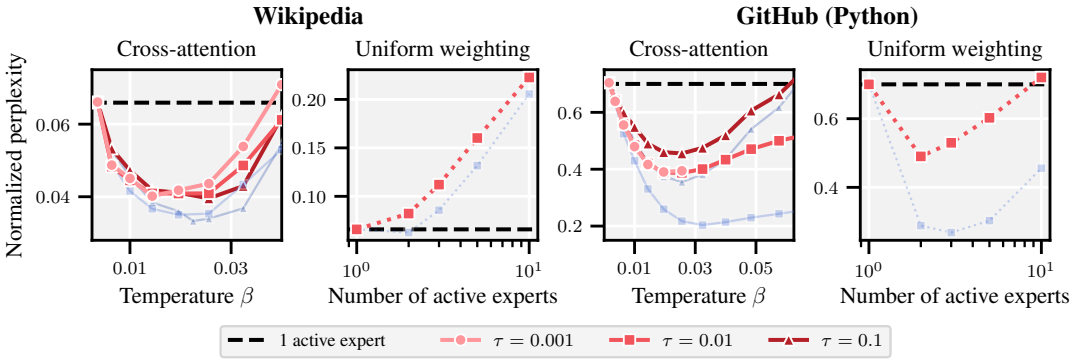

Figure 5: On each dataset, we evaluate the performance improvement of TTMM's cross-attention mechanism compared to uniform weighting of active experts. Uniform weighting (dotted) selects the experts closest to the prompt embedding, and weights them equally. We see across merging (red) and ensembling (light blue) that the cross-attention mechanism outperforms uniform weighting significantly. Notably, on Wikipedia, combining multiple experts helps only when using cross-attention. The number of experts is $K = 1000$.

Insight 1: TTMM approaches the accuracy of TTT at almost no test-time overhead. We show in Figure 1 and Table 1 that TTMM with 10 active experts achieves close to the accuracy of TTT, and much higher accuracy than a single model that was fine-tuned on the respective corpus. Instead of merging experts in parameter space, we also evaluate merging experts in prediction space by ensembling the experts. We find that this yields marginally higher accuracy, yet it has a significantly larger test-time overhead. Overall, merging experts in parameter space achieves the best trade-off between accuracy and test-time efficiency. Interestingly, TTMM outperforms TTT on the Wikipedia dataset. This may be because TTMM can incorporate knowledge from a larger set of experts, each trained on a few thousands of data points. In contrast, TTT is only training on the 100 most related neighbors, which may miss some information if there are more than 100 related data points. In our view, the empirical difference between TTMM and TTT is an interesting direction for further study.

Insight 2: Cross-attention is more effective than other merging coefficients. In Figure 5, we evaluate perplexity with merging coefficients computed via cross-attention to uniform

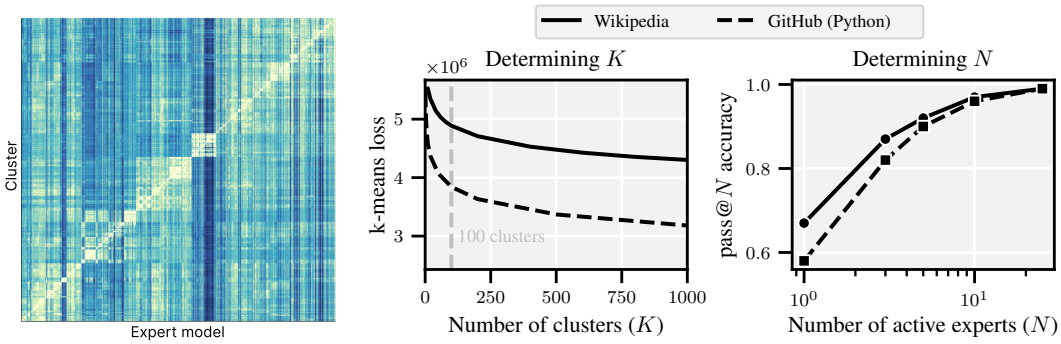

Figure 6: **Left:** Perplexity (brighter is better) of 1000 experts on clusters of the Wikipedia dataset. The lowest perplexity is achieved on the diagonal, i.e., when the expert model is evaluated on holdout data from the cluster it was trained on. The block-diagonal structure is an artifact of bisecting k-means, which leads to similar neighboring clusters. **Middle:** In our evaluation datasets, the rate of reduction in k-means loss is largest until around 100 clusters. **Right:** For 100 random samples from each cluster's holdout set, we measure whether the correct cluster is among the top-N closest clusters and report the average pass@N accuracy.

weighting of active experts. We find that cross-attention significantly outperforms uniform weighting. In particular, on the Wikipedia dataset, merging multiple experts does not help at all when weighting them uniformly. We also compare against merging coefficients, such as leveraging the logit-entropy of the experts (cf. Appendix D.3), but we do not find them to outperform TTMM's cross-attention while being significantly more computationally expensive. We find that sparsity $\tau = 0.01$ provides the best trade-off between accuracy and test-time efficiency, and we fix it in all other experiments. Furthermore, on our evaluation corpora, we need to select roughly 10 active experts to cover the local neighborhood of the prompt (cf. Figure 6 (right)), which is also when the performance of TTMM begins to saturate.

Insight 3: Local experts outperform the base model in their local neighborhood. We find in Figure 6 (left) that expert models perform significantly better on holdout data in their cluster than the base model or other expert models. This reaffirms the earlier findings of Hardt & Sun (2024) that TTT on local neighborhoods improves accuracy. We also find a large qualitative difference between the text generated by different experts and the base model, and we include examples in Appendix F.

Choosing the number of (active) experts. We find, as summarized in Figure 1, that TTMM with 100 experts significantly outperforms TTMM with 10 experts or the multitask fine-tuned model (i.e., a single expert). This indicates that scaling the number of experts beyond the number of meta-tasks can increase accuracy substantially. Nevertheless, we also find that scaling the number of experts to 1000 does not further improve accuracy on our evaluation corpora. In the following, we discuss considerations for the number of experts in TTMM:

- **Number of experts (K):** The total number of experts has almost no direct effect on latency. However, more experts require a large amount of static CPU memory. Further, increasing the number of experts also increases the separation of knowledge/skills between experts. This then requires more active experts to retain the same performance, which also increases latency and may lead to model interference due to merging. To prevent knowledge fragmentation, K should be determined depending on the training set. In our experiments, choosing 100 experts is indicated by the "elbow method", common in clustering (cf. Figure 6 (middle)).
- **Number of active experts (N):** The number of active experts has the strongest effect on latency, since all active experts have to be loaded from CPU into GPU memory before inference. Our ablation in Figure 6 (right) indicates that with 10 active experts, the "correct" expert is likely to be active, while latency remains minimal.

Evaluation on MMLU. We additionally evaluate TTMM on MMLU (Hendrycks et al., 2021) as a non-perplexity task. We use (non-instruct) Llama-3.2-1B as base model, which we then fine-tune for one epoch on the MMLU training set. This fine-tuned model achieves approx-

Model	Humanities	Social Sciences	STEM	Other	Overall
Fine-tuned	44.8	54.24	40.79	54.43	48.10
TTMM (1 expert)	45.25	54.92	40.98	54.30	48.41
TTMM (3 experts)	45.33	55.02	40.98	54.39	48.48
TTMM (10 experts)	**45.63**	55.15	41.39	54.55	48.74
TTMM (15 experts)	45.46	**55.31**	**41.77**	**55.26**	**48.96**

Table 2: Accuracy (in %) of TTMM on MMLU. **Bold** numbers denote the highest accuracy.

imately the same performance as instruction-tuned Llama-3.2-1B. We use the non-instruct model to avoid any confounding effect of RL-training or instruction-tuning on privileged data from the instruct model. As embedding model, we use Stella (`stella_en_400M_v5`) but obtain similar results with MPNet (cf. Appendix D.2). We train 100 experts on the MMLU training set, using the globally fine-tuned model as initialization. We summarize the results in Table 2 and find that TTMM slightly increases accuracy. A broader evaluation on other non-perplexity tasks and non-language domains is an important direction for future work.

Ablations. We perform a series of ablations, which we summarize in Appendix D. Notably, we find that (1) larger embedding models need not lead to improved performance of TTMM and that (2) TTMM performs similarly when fixing the number of experts as to when selecting the number of active experts dynamically via thresholding.

5 Discussion and Future Work

We propose ***T**est-**T**ime **M**odel **M**erging* (TTMM) which scales the mixture of experts paradigm to orders of magnitude more experts than in state-of-the-art multitask methods, leading to a substantial improvement in language modeling ability. We find that TTMM can achieve performance gain close to TTT by merging *few* active experts into a single model per query. TTMM achieves this with almost the same test-time cost as inference with the base model, and without increasing train-time FLOPs or GPU memory compared to multitask fine-tuning. A limitation of TTMM is the additional CPU memory required for storing all experts.

In achieving close to the performance of TTT without nearly any test-time overhead, TTMM opens up several exciting directions for future research. First, TTMM makes it feasible to re-select models online during generation or evaluation based on the growing prefix, which with TTT, is too expensive in most applications. Gururangan et al. (2023) have already shown that online re-selection can work with an ensemble selected from a smaller number of experts. Second, one may further improve the performance of TTMM using tools from the literature on model merging and MoEs aimed at avoiding interference between experts and improving the routing of data to experts, e.g., by fine-tuning the router on holdout data. TTMM can further be applied to settings with privileged information by adding or removing experts dynamically based on user access. Furthermore, it may be possible to leverage the large number of experts in TTMM for uncertainty estimation, e.g., by tracking also the variance of the merged parameters. Finally, many reasoning language models (called RLMs) such as DeepSeek R1 (Guo et al., 2025) are based on a MoE architecture, and TTT has been shown to be able to improve RLMs (Zuo et al., 2025). Since TTMM can be considered a scalable MoE architecture that approximates TTT, it would be particularly interesting to study whether TTMM can improve reasoning models.

Acknowledgments

This project was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and Innovation Program Grant agreement no. 815943, and the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40 180545. Ido Hakimi was supported by an ETH AI Center Postdoctoral fellowship.

References

Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *ICLR*, 2023.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot-learning. In *ICML*, 2025.

Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning. *Lazy learning*, 1997.

Marco Bagatella, Jonas Hübotter, Georg Martius, and Andreas Krause. Active fine-tuning of multi-task policies. In *ICML*, 2025.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *ICML*, 2022.

Léon Bottou and Vladimir Vapnik. Local learning algorithms. *Neural computation*, 4(6), 1992.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint ArXiv:2005.14165*, 2020.

Feng Cheng, Ziyang Wang, Yi-Lin Sung, Yan-Bo Lin, Mohit Bansal, and Gedas Bertasius. Dam: Dynamic adapter merging for continual video qa learning. *arXiv preprint arXiv:2403.08755*, 2024.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *ICML*, 2022.

William S Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368), 1979.

William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403), 1988.

Nico Daheim, Thomas Möllenhoff, Edoardo Maria Ponti, Iryna Gurevych, and Mohammad Emtiyaz Khan. Model merging by uncertainty-based gradient matching. In *ICLR*, 2024.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Karan Dalal, Daniel Koceja, Gashon Hussein, Jiarui Xu, Yue Zhao, Youjin Song, Shihao Han, Ka Chun Cheung, Jan Kautz, Carlos Guestrin, et al. One-minute video generation with test-time training. *arXiv preprint arXiv:2504.05298*, 2025.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, 2000.

Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei Efros. Test-time training with masked autoencoders. In *NeurIPS*, 2021.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Suchin Gururangan, Margaret Li, Mike Lewis, Weijia Shi, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Scaling expert language models with unsupervised domain discovery. *arXiv preprint arXiv:2303.14177*, 2023.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *ICML*, 2020.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *ICLR*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*, 2021.

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 *ICLR*, 2022.

Jonas Hübotter, Lenart Treven, Yarden As, and Andreas Krause. Transductive active learning: Theory and applications. In *NeurIPS*, 2024.

Jonas Hübotter, Sascha Bongni, Ido Hakimi, and Andreas Krause. Efficiently learning at test-time: Active fine-tuning of llms. In *ICLR*, 2025.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *ICLR*, 2023.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018.

Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8), 2010.

Vidit Jain and Erik Learned-Miller. Online domain adaptation of a pre-trained cascade of classifiers. In *CVPR*, 2011.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *ACL*, 2023.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *ICLR*, 2023.

Aecheon Jung, Seunghwan Lee, Dongyoon Han, and Sungeun Hong. Why train everything? tint a single layer for multi-task model merging. *arXiv preprint arXiv:2412.19098*, 2024.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *ICLR*, 2020.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. In *ICML*, 2018.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of transformer language models. *arXiv preprint arXiv:1904.08378*, 2019.

Kunfeng Lai, Zhenheng Tang, Xinglin Pan, Peijie Dong, Xiang Liu, Haolan Chen, Li Shen, Bo Li, and Xiaowen Chu. Mediator: Memory-efficient llm merging with less parameter conflicts and uncertainty based routing. *arXiv preprint arXiv:2502.04411*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*, 2019.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A Smith, and Yejin Choi. Dexperts: Decoding-time controlled text generation with experts and anti-experts. In *ACL*, 2021.

David Lowe and D Broomhead. Multivariable functional interpolation and adaptive networks. *Complex systems*, 2(3), 1988.

Zhenyi Lu, Chenghao Fan, Wei Wei, Xiaoye Qu, Dangyang Chen, and Yu Cheng. Twin-merging: Dynamic integration of modular expertise in model merging. In *NeurIPS*, 2024.

Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. In *NeurIPS*, 2022.

Costas Mavromatis, Petros Karypis, and George Karypis. Pack of llms: Model fusion at test-time via perplexity optimization. In *COLM*, 2024.

John Moody and Christian J Darken. Fast learning in networks of locally-tuned processing units. *Neural computation*, 1(2), 1989.

Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1), 1964.

Changdae Oh, Yixuan Li, Kyungwoo Song, Sangdoo Yun, and Dongyoon Han. Dawin: Training-free dynamic weight interpolation for robust adaptation. In *ICLR*, 2025.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3), 1962.

A Paszke. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Zihuan Qiu, Yi Xu, Chiyuan He, Fanman Meng, Linfeng Xu, Qingbo Wu, and Hongliang Li. Mingle: Mixtures of null-space gated low-rank experts for test-time continual model merging. *arXiv preprint arXiv:2505.11883*, 2025.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *IJCNLP*, 2019.

Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3), 1956.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *CVPR*, 2023.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

Assaf Shocher, Nadav Cohen, and Michal Irani. "zero-shot" super-resolution using deep internal learning. In *CVPR*, 2018.

Toby Simonds and Akira Yoshiyama. Ladder: Self-improving llms through recursive problem decomposition. *arXiv preprint arXiv:2503.00735*, 2025.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. In *NeurIPS*, 2020.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, et al. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. *arXiv preprint arXiv:2403.07816*, 2024.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Anke Tang, Li Shen, Yong Luo, Nan Yin, Lefei Zhang, and Dacheng Tao. Merging multi-task models via weight-ensembling mixture of experts. In *ICML*, 2024a.

Anke Tang, Li Shen, Yong Luo, Yibing Zhan, Han Hu, Bo Du, Yixin Chen, and Dacheng Tao. Parameter efficient multi-task model fusion with partial linearization. In *ICLR*, 2024b.

Anke Tang, Enneng Yang, Li Shen, Yong Luo, Han Hu, Bo Du, and Dacheng Tao. Merging models on the fly without retraining: A sequential approach to scalable continual model merging. *arXiv preprint arXiv:2501.09522*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.

Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, 1964.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Wikimedia Foundation. Wikimedia downloads, 2025. URL `https://dumps.wikimedia.org`.

Andrew Gordon Wilson. Deep learning is not so mysterious or different. *arXiv preprint arXiv:2503.02113*, 2025.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. In *NeurIPS*, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *arXiv preprint arXiv:2408.07666*, 2024b.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *ICLR*, 2024c.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *ICML*, 2024.

Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. Jasper and stella: distillation of sota embedding models. *arXiv preprint arXiv:2412.19048*, 2024a.

Qizhen Zhang, Nikolas Gritsch, Dwaraknath Gnaneshwar Talupuru, Simon Guo, David Cairuz, Bharat Venkitesh, Jakob Foerster, Phil Blunsom, Sebastian Ruder, Ahmet Üstün, et al. Bam! just like that: Simple and efficient parameter upcycling for mixture of experts. In *NeurIPS*, 2024b.

Qizhen Zhang, Prajjwal Bhargava, Chloe Bi, Chris X Cai, Jakob Foerster, Jeremy Fu, Punit Singh Koura, Ruan Silva, Sheng Shen, Emily Dinan, et al. Bts: Harmonizing specialized experts into a generalist llm. *arXiv preprint arXiv:2502.00075*, 2025.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. In *NeurIPS*, 2022.

Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.

Appendices

```
for name in lora_param_names:
  As = torch.stack([
    d[name]["A"] for d in lora_dicts])
  Bs = torch.stack([
    d[name]["B"] for d in lora_dicts])
  delta = torch.einsum(
    "k i r, k r o -> i o", Bs, As)
```

Figure 7: We merge LoRA adapters using PyTorch's `einsum`, before applying the low-rank updates to the model parameters.

A Additional Related Work

Retrieval-Augmented Generation. An alternative to specializing model parameters at test-time (i.e., TTT) is to instead specialize predictions by conditioning an autoregressive model directly on prompt-related data. This is called retrieval-augmented generation (RAG, Lewis et al., 2019; Khandelwal et al., 2020; Guu et al., 2020; Borgeaud et al., 2022) and motivated by the ability of LLMs to learn from context (Brown et al., 2020; Wei et al., 2022). Similarly to TTT, RAG leads to expensive inference since all local data needs to be processed at test-time. TTMM is more efficient by compressing local information into expert LoRA adapters at train-time. RAG is commonly used to endow LLMs with privileged information. Similarly, TTMM can include (& remove) information simply by adding (or removing) local experts.

B Equivalence of Cross-Attention and RBF Kernel

Assuming that all embeddings are normalized, we have

$$\exp\left(-\frac{1}{2\beta}\|\boldsymbol{\phi}^\star - \boldsymbol{\phi}_k\|_2^2\right) = \exp\left(-\frac{1}{\beta} + \frac{1}{\beta}\boldsymbol{\phi}_k^\top \boldsymbol{\phi}^\star\right) \propto \exp\left(\frac{1}{\beta}\boldsymbol{\phi}_k^\top \boldsymbol{\phi}^\star\right).$$

C Proof of Informal Proposition 3.1

Setup. Let $f(\boldsymbol{x};\boldsymbol{\theta})$ be a neural network parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$ and evaluated on input $\boldsymbol{x} \in \mathcal{X}$. We have a loss function $\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}) \in \mathbb{R}$ for data point $\boldsymbol{z} = (\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$.[3] Let $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ be a dataset of input-output pairs. Given a prompt $\boldsymbol{x}^\star \in \mathcal{X}$ and any $N \geq 1$, we consider the set of N nearest neighbors to $\boldsymbol{\phi}(\boldsymbol{x}^\star)$ in $\mathcal{D}$ (with respect to the input embedding):

$$\mathcal{D}_{\boldsymbol{x}^\star} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\} \subseteq \mathcal{D}.$$

Let $\mathcal{D}'$ be any dataset of size $M \geq 1$:

$$\mathcal{D}' = \{\boldsymbol{z}'_1, \ldots, \boldsymbol{z}'_M\} \subseteq \mathcal{D}.$$

We first analyze single-step gradient descent with step size $\eta > 0$ from the same initialization $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_{\boldsymbol{x}^\star} = \boldsymbol{\theta} - \eta\left[\frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}_i)\right], \quad \boldsymbol{\theta}' = \boldsymbol{\theta} - \eta\left[\frac{1}{M}\sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}'_j)\right].$$

Assumptions.

(A1) **Lipschitz-smoothness of the loss.** There is a constant $G > 0$ such that for all $\boldsymbol{z} = (\boldsymbol{x}, y), \boldsymbol{z}' = (\boldsymbol{x}', y')$ and all $\boldsymbol{\theta}$:

$$\left\|\nabla_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}) \;-\; \nabla_{\boldsymbol{\theta}}\,\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{z}')\right\| \;\leq\; G\,\|\boldsymbol{x} - \boldsymbol{x}'\|.$$

[3] This supervised data may be constructed via self-supervision, as in language modeling.

(A2) **Network Lipschitz in parameters.** There is a constant $L > 0$ such that for all $\boldsymbol{\theta}, \boldsymbol{\theta}'$ and all $\boldsymbol{x}$:

$$\left\| f(\boldsymbol{x}; \boldsymbol{\theta}) \; - \; f(\boldsymbol{x}; \boldsymbol{\theta}') \right\| \; \leq \; L \, \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|.$$

Proposition C.1. *Suppose (A1) and (A2) hold, and let $\boldsymbol{\theta}$ be any initialization. For any prompt $\boldsymbol{x}^\star \in \mathcal{X}$ and $N \geq 1$, let $\boldsymbol{\theta}_{\boldsymbol{x}^\star}$ be the model trained via single-step gradient descent with step size $\eta > 0$ on $\mathcal{D}_{\boldsymbol{x}^\star}$. Let $\boldsymbol{\theta}'$ be the expert model trained on any $\mathcal{D}' \subseteq \mathcal{D}$. Then, if $\mathcal{D}' \cap \mathcal{D}_{\boldsymbol{x}^\star} \neq \emptyset$,*

$$\left\| f(\boldsymbol{x}^\star; \boldsymbol{\theta}_{\boldsymbol{x}^\star}) - f(\boldsymbol{x}^\star; \boldsymbol{\theta}') \right\| \leq \eta L G (\operatorname{diam}(\mathcal{D}_{\boldsymbol{x}^\star}) + \operatorname{diam}(\mathcal{D}')). \tag{2}$$

Proof. We have

$$\begin{aligned}
\boldsymbol{\theta}_{\boldsymbol{x}^\star} - \boldsymbol{\theta}' &= -\eta \left[\frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z_i) - \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z'_j) \right] \\
&= -\eta \left[\frac{1}{M} \sum_{j=1}^{M} \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z_i) - \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z'_j) \right] \\
&= -\eta \left[\frac{1}{M} \sum_{j=1}^{M} \left(\frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z_i) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z'_j) \right) \right].
\end{aligned}$$

Thus,

$$\begin{aligned}
\|\boldsymbol{\theta}_{\boldsymbol{x}^\star} - \boldsymbol{\theta}'\| &\leq \eta \frac{1}{M} \sum_{j=1}^{M} \frac{1}{N} \sum_{i=1}^{N} \|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z_i) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; z'_j)\| && \text{(triangle inequality)} \\
&\leq \eta G \frac{1}{M} \sum_{j=1}^{M} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{x}'_j\|. && \text{(Assumption A1)}
\end{aligned}$$

Let $\boldsymbol{x}$ be any point in $\mathcal{D}' \cap \mathcal{D}_{\boldsymbol{x}^\star}$. We have

$$\begin{aligned}
&= \eta G \frac{1}{M} \sum_{j=1}^{M} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{x} + \boldsymbol{x} - \boldsymbol{x}'_j\| \\
&\leq \eta G \frac{1}{M} \sum_{j=1}^{M} \frac{1}{N} \sum_{i=1}^{N} \Big(\|\boldsymbol{x}_i - \boldsymbol{x}\| + \|\boldsymbol{x} - \boldsymbol{x}'_j\| \Big) && \text{(triangle inequality)} \\
&\leq \eta G (\operatorname{diam}(\mathcal{D}_{\boldsymbol{x}^\star}) + \operatorname{diam}(\mathcal{D}')).
\end{aligned}$$

Finally,

$$\begin{aligned}
\|f(\boldsymbol{x}^\star; \boldsymbol{\theta}_{\boldsymbol{x}^\star}) - f(\boldsymbol{x}^\star; \boldsymbol{\theta}')\| &\leq L \|\boldsymbol{\theta}_{\boldsymbol{x}^\star} - \boldsymbol{\theta}'\| && \text{(Assumption A2)} \\
&\leq \eta L G (\operatorname{diam}(\mathcal{D}_{\boldsymbol{x}^\star}) + \operatorname{diam}(\mathcal{D}')).
\end{aligned}$$

□

To complete the proof of Informal Proposition 3.1, we simply observe that the bound $\|\boldsymbol{\theta}^{(1)}_{\boldsymbol{x}^\star} - \boldsymbol{\theta}'^{(1)}\| \leq \eta G (\operatorname{diam}(\mathcal{D}_{\boldsymbol{x}^\star}) + \operatorname{diam}(\mathcal{D}'))$ can be applied recursively from the shared initialization $\boldsymbol{\theta}^{(0)}$ for T steps of gradient descent via the relation

$$\|\boldsymbol{\theta}^{(t+1)}_{\boldsymbol{x}^\star} - \boldsymbol{\theta}'^{(t+1)}\| \leq \|\boldsymbol{\theta}^{(t)}_{\boldsymbol{x}^\star} - \boldsymbol{\theta}'^{(t)}\| + \eta G (\operatorname{diam}(\mathcal{D}_{\boldsymbol{x}^\star}) + \operatorname{diam}(\mathcal{D}')).$$

Unrolling this recursion proves Informal Proposition 3.1 with an additional factor T compared to Proposition C.1.

Model	# Active Experts	Wikipedia		GitHub (Python)	
		TTMM	Fixed	TTMM	Fixed
Llama-3.2-1B	1	7.669	7.669	2.552	2.552
	3	7.571	7.580	2.519	2.505
	10	7.510	7.509	2.492	2.491
Qwen2.5-1.5B	1	7.166	7.166	2.330	2.331
	3	7.101	7.104	2.304	2.295
	10	7.080	7.080	2.287	2.287

Table 3: Perplexity comparison (lower is better) between TTMM (dynamic expert routing) and fixed expert selection for Llama and Qwen across different numbers of active experts.

Model	Humanities	Social Sciences	STEM	Other	Overall
Fine-tuned	44.80	54.24	40.79	54.43	48.10
`all-mpnet-base-v2`	**45.95**	54.76	40.56	54.72	48.61
`stella_en_400M_v5`	45.46	**55.31**	**41.77**	**55.26**	**48.96**
`gte-Qwen2-1.5B-instruct`	45.27	54.76	40.91	54.65	48.45

Table 4: MMLU accuracy (in %) comparing TTMM with different embedding models against the fine-tuned baseline. **Bold** indicates the highest accuracy.

D Ablations

D.1 Fixing the Number of Active Experts

In Table 3, we ablate the choice of selecting the number of active experts dynamically per prompt depending on the threshold τ (cf. Algorithm 2) rather than specifying a fixed sparsity N. On our evaluated corpora, we find no significant difference in performance.

D.2 Different Embedding Models

In Table 4, we evaluate different embedding models on MMLU and find that TTMM outperforms the base model regardless of the choice of embedding model. Notably, the largest Qwen2-1.5B embedding model performs worse than the 100M MPNet model. A possible reason is that finding a decent clustering might be significantly easier than directly solving questions, since the former only requires a broad understanding of the meaning of some keywords while the latter requires some ability of reasoning. Stella (Zhang et al., 2024a) performs the best among the tested embedding models.

D.3 Evaluation of Alternative Merging Coefficients

SIFT: Accounting for redundancy between experts. We evaluate weighting active experts according to their estimated uncertainty reduction about the response to the prompt, as proposed by Hübotter et al. (2025). Building on transductive active learning (Hübotter et al., 2024), Hübotter et al. (2025) approximate the expected reduction in uncertainty about the prompt when conditioning on the i-th data point by $\sigma_{i-1}^2 - \sigma_i^2$ where σ_i^2 estimates the uncertainty after seeing the i-th data point.

We use the same uncertainty estimates σ_i^2 to weight the experts in TTMM, but instead of conditioning on individual data points represented by their embeddings, we condition on expert models represented by their centroids. For a given prompt, we first retrieve the N

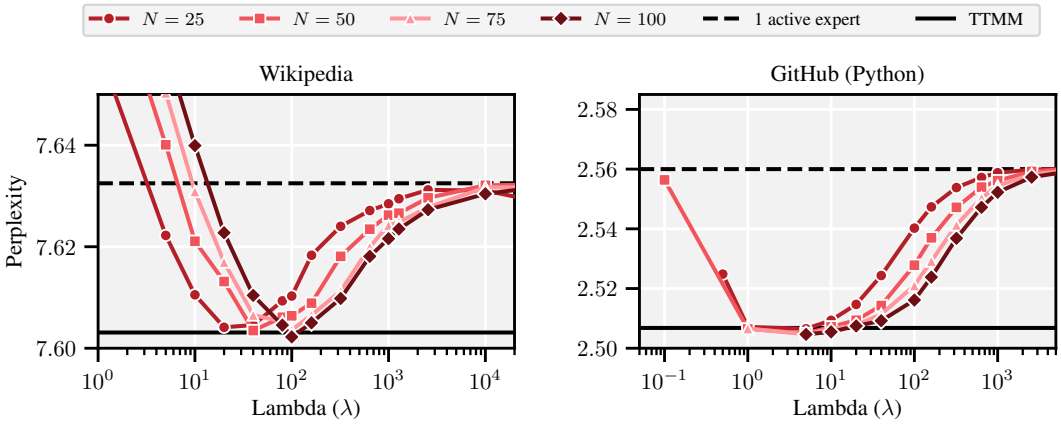

Figure 8: We evaluate weighting experts according to SIFT's uncertainty estimates, which take into account redundancy between clusters. Here, λ is the hyperparameter of SIFT. As $\lambda \to \infty$, SIFT selects only the centroid closest to the prompt embedding, and as $\lambda \to 0$, SIFT weights all clusters uniformly.

nearest centroids and then reweight them according to:

$$w_k \leftarrow \frac{\sigma_{k-1}^2 - \sigma_k^2}{\sum_{k'=1}^{N} \sigma_{k'-1}^2 - \sigma_{k'}^2} = \frac{\sigma_{k-1}^2 - \sigma_k^2}{\sigma_0^2 - \sigma_N^2}.$$

Given that embeddings are normalized, we have $\sigma_0^2 = 1$.

Intuitively, these weights account for redundancy between clusters. Suppose we duplicate a cluster m times, then the above weights downweigh each of the m duplicates by a factor $1/m$. The cross-attention mechanism in TTMM, on the other hand, would treat each duplicate cluster as if it were a distinct cluster, and hence increase the weight of the duplicated cluster by m.

We evaluate SIFT in Figure 8. We find that when we have control over the created experts (e.g., by clustering the data) as in TTMM, SIFT yields only a negligible improvmenet over the cross-attention mechanism. This likely stems from the fact that the centroids of the clusters are already well-separated, and hence the cross-attention mechanism is able to effectively weight the experts. We expect that in settings where we have less control over the experts, SIFT-weighting could provide a more significant improvement.

DaWin: Weighting experts by their logit-entropy. Oh et al. (2025) propose DaWin, which weights experts by their logit-entropy. Let $p_k(\boldsymbol{x}^\star)$ be the distribution over logits of expert k for input $\boldsymbol{x}^\star$ with $H(p_k(\boldsymbol{x}^\star))$ denoting its entropy. Then, the weight of expert k is given by

$$w_k \leftarrow \text{sparse-softmax}_\tau\left(-\frac{1}{\beta} H(p_k(\boldsymbol{x}^\star))\right).$$

Note that these weights are equivalent to the weights obtained by the cross-attention mechanism in TTMM, up to the use of $-H(p_k(\boldsymbol{x}^\star))$ to measure similarity as opposed to $\boldsymbol{\phi}_k^\top \boldsymbol{\phi}(\boldsymbol{x}^\star)$. As opposed to the inner product, which can be evaluated with very little overhead, computing the entropy requires a separate forward pass with each expert. This makes it as expensive as ensembling experts at test-time. In Figure 9, we compare the logit-entropy weighting to the inner product weighting of TTMM, and do not find a performance difference.

Ablation against summarizing each expert by its clusters' centroid. As suggested in Section 3.2, we evaluate whether summarizing each expert by its clusters' centroid meaningfully degenerates the performance of TTMM. We find that this appears not to be the case, as shown in Table 5.

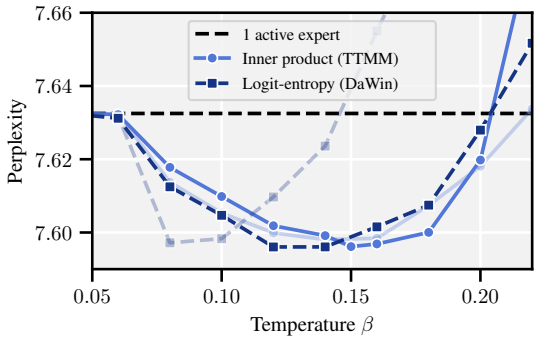

Figure 9: We evaluate ensembling models with inner product weighting (like in TTMM) and logit-entropy weighting (like in DaWin). We evaluate on Wikipedia with Llama-3.2-1B as base model. Solid lines are with sparsity $\tau = 0.1$, translucent lines are with $\tau = 0.01$.

	Selection criterion	**Perplexity (lower is better)**
Summarized by centroid	$\|\boldsymbol{\phi}^\star - \boldsymbol{\phi}_k\|^2$	7.669
Not summarized by centroid	$\sum_{\boldsymbol{x}\in\mathcal{D}_k} \|\boldsymbol{\phi}^\star - \boldsymbol{\phi}(\boldsymbol{x})\|^2$	7.633

Table 5: We evaluate the selection of experts, without approximating each expert by its clusters' centroid as suggested in Section 3.2. The evaluation is on the Wikipedia corpus with Llama-3.2-1B as base model, $K = 1000$ total experts and a single active expert.

E Experiment Details

Parameter	**Value**
Number of clusters K	100
Learning rate	2e-4
LoRA rank	64
LoRA alpha	16
Adam epsilon	1e-8
Adam beta	(0.9, 0.999)
Weight decay	0.01
Batch size	4
Sparsity τ	0.01

Table 6: Hyperparameters.

Dataset	**Source**
Wikipedia	`https://huggingface.co/datasets/wikimedia/wikipedia`
GitHub (filtered to Python)	`https://huggingface.co/datasets/codeparrot/github-code-clean`

Table 7: Datasets.

All models were trained on the first 1024 tokens of each training example. Testing was conducted on the full input length, with the context size limited to 2^{14} tokens. For each dataset, the test sets were constructed by selecting a single random example from each cluster's holdout set, ensuring that the evaluation approximately covers the data of all clusters.

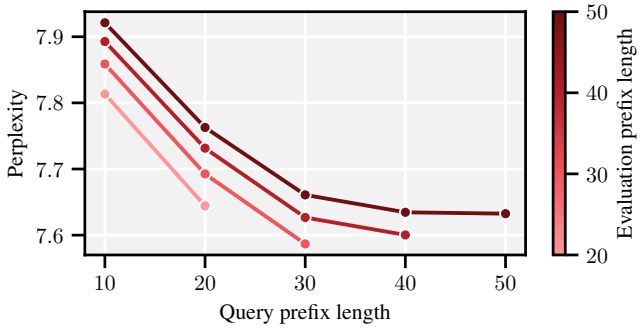

Figure 10: On Wikipedia and with Llama-3.2-1B as base model, we evaluate different configurations of query prefix length (the number of tokens used as query) and evaluation prefix length (the number of tokens ignored for evaluation). Around 40 to 50 tokens, the performance gain of including more tokens in the query begins to diminish.

The hyperparameters for TTT match those used to train the expert models, except for a learning rate of 5e-4. The models were trained for five epochs and the best checkpoint was selected for evaluation.

Unless noted otherwise, all ablations are with Llama-3.2-1B as the base model. Further, unless noted otherwise, we use "number of active experts" as an upper bound to the mean number of active experts across test examples.

Dataset size for expert training. Both the Wikipedia and the GitHub (Python) corpora contain roughly 7M documents. Therefore, since each train document is at most 1024 tokens long, each expert is trained on at most ≈7M tokens if we train 1000 experts. This is a relatively small dataset size for training experts, compared to datasets used for training common MoE or multitask models.

Preventing Information Leakage. The TTT setting (and also TTMM) requires specification of a prompt to obtain a task-specific model. In language modeling datasets that do not have the natural structure of prompt and response, it would be unfair to use the response as prompt. Using the response as prompt would invalidate the evaluation, as the test data is leaked to the model before its evaluation. To prevent this, we split individual data points into a prefix (prompt) and suffix (response) similar to Hardt & Sun (2024), use the prefix for obtaining the task-specific model, and evaluate the model on the suffix. Table 8 summarizes the choice of prefix per dataset. When not dynamically re-selecting models before the generation of each token (or after every N tokens), the prefix must be sufficiently informative for the suffix. On Wikipedia, we find this to be around 50 tokens (cf., Figure 10).

	Prefix length (in tokens)
Wikipedia	50
GitHub (Python)	200

Table 8: Overview of prefix lengths used per evaluation dataset.

F Qualitative Examples

1. **Clustering:** To investigate the clustering, we use an LLM to summarize each cluster's content with a title based on 10 data points of each cluster. Tables 9 and 10 list some of those cluster titles.
2. **Expert Selection:** Table 11 shows examples of selected experts for different queries.

3. **Model Generation:** We find that expert models generate distinctly different text. Table 12 gives examples of generations from empty strings, while Table 13 shows examples of generations from prompts. Within their subject of expertise, expert models tend to hallucinate less or at a later stage than the base model.

Cluster number	Cluster title
118	Aviation History and Aviation-Related Topics
119	Historical Figures and Notable Americans
120	Historical Figures and Events in the United States
612	Honey Bees and their Scientific Classification
613	Small Mammals and Insects
614	Insect Taxonomy and Systematics
968	Swiss Geography and Mountains
969	Rivers of Germany
970	Historical Places and Landmarks in Europe
971	German Municipalities and Towns

Table 9: Wikipedia cluster titles

Cluster number	Cluster title
17	Kivy GUI Framework
18	Tkinter GUI Framework Development
19	Multi-Touch Plugin for Alignak
427	Hidden Markov Model Implementation
428	Sparse Matrix Decomposition and Optimization
429	Logistic Regression with L2 Regularization
513	3D Printing and CAD Design
514	3D Mesh Generation and Manipulation Tools
979	Database Schema Definition and Migration Tools
980	MongoDB Database Schema Management

Table 10: GitHub (Python) cluster titles

Query input	Closest expert	2nd-closest expert	3rd-closest expert
Basketball is a great sport	**(309)** Women's Basketball Competitions and Leagues	**(281)** NBA and WNBA Teams and Seasons	**(311)** Sports and Athletics
Albert Einstein was an astounding physicist	**(169)** Historical Physicists and Scientists with Nobel prizes	**(104)** Tufts University Alumni and Faculty in Science	**(88)** British Scientists and Mathematicians
UNESCO World Heritage Site	**(716)** Cities and Urban Landmarks	**(752)** Asian Landmarks and Cultural Sites	**(906)** Places in South Africa

Table 11: Example Wikipedia expert selection

Model	Generated text
Base Model	**Ex 1.** Question: What is the greatest common factor of 30 and 15? Answer: 15 **Ex 2.** Question: Design a Python function snippet to Check High Handling personal items: Sanitizing Frequently Touched Items for Analysis for Beginners. Implement if/else or switch/case statements to handle different conditions related to the Reliability. Provide detailed comments explaining your control flow and the reasoning behind each decision.
Expert **(1)**: Politicians and Political Parties from Southeast Asia	**Ex 1.** The 2018 South Korean by-elections was held in South Korea on 9 January 2018. The by-elections were held to fill two vacant seats in the National Assembly. The first by-election was held for the vacant seat in the constituency of Busan 1st District, and the second for the vacant seat in Seoul 6th District. **Ex 2.** The National Congress of Australia's First Peoples (NCAFP) is an organisation that advocates for the interests of Aboriginal and Torres Strait Islander Australians in the Australian Parliament. It is a member of the Council of Australian Traditional Owner's Organisations (CATOO).
Expert **(280)**: American Football Teams and Players	**Ex 1.** The 2017 season was the New England Patriots' 48th in the National Football League (NFL), their 58th overall, their 51st playing home games at Gillette Stadium and their sixth under head coach Bill Belichick. The Patriots entered the season as the defending Super Bowl champions and were looking to defend their AFC East division title and make it to their third consecutive Super Bowl. **Ex 2.** The 2006 Houston Texans season was the franchise's 7th season in the National Football League (NFL) and the 1st under head coach Gary Kubiak. They improved on their 4–12 record from 2005 and made the playoffs for the first time since 2003. They won the AFC South for the first time since 2002 and defeated the Indianapolis Colts in the Wild Card round 31–17.

Table 12: Generation from empty string inputs

Model	Generated text
Base Model	**Ex 1.** **The politics in** the US, the UK and elsewhere have been dominated by the "Brexit" vote to leave the European Union. While many people in the UK and elsewhere have been unhappy with the outcome, it is worth remembering that the vote was very close. The referendum was held on June 23rd, 2016, and the result was 52% in favour of leaving and 48% in favour of staying. **Ex 2.** **Super Bowl** 50: 5 Things to Know The 2016 Super Bowl will be played on February 7, 2016. It will be the 50th Super Bowl and the second to be played in the new 20,000 seat Levi's Stadium in Santa Clara, California. Here are five things to know about the Super Bowl 50: 1. The Super Bowl is played on the first Sunday in February each year. The last Super Bowl was played on February 7, 2015.
Expert **(1)**: Politicians and Political Parties from Southeast Asia	**Ex 1.** **The politics in** the Cook Islands are characterised by a multi-party system and an electoral calendar dominated by the 2018 general election. The Cook Islands has had a democratic political system since 1965, with the Cook Islands being the first country in the Pacific to hold elections for the Parliament. **Ex 2.** **Super Bowl** XLIV was an American football game that was held on Sunday, February 7, 2010, at Superdome, New Orleans, Louisiana. The 2010 Super Bowl was the 45th edition of the American Football League Championship Game and the 46th overall Super Bowl.
Expert **(280)**: American Football Teams and Players	**Ex 1.** **The politics in** the 1990 NFL season involved the 1990 NFL Draft, the 1990 Pro Bowl, and the 1990 NFL season ending in controversy over the 1990 NFC Championship Game. The 1990 season was the first NFL season in which the league was divided into two conferences, with the NFC and AFC, and the first to end in controversy. **Ex 2.** **Super Bowl** LIV was an American football game played to determine the champion of the National Football League (NFL) for the 2019 season. The American Football Conference (AFC) champion Kansas City Chiefs defeated the National Football Conference (NFC) champion San Francisco 49ers, 31–20, on February 2, 2020, at Hard Rock Stadium in Miami Gardens, Florida.

Table 13: Generation from input strings, input is in **bold**