Superposition in Transformers: A Novel Way of Building Mixture of Experts

Ayoub Ben Chaliah
ayoub1benchaliah@gmail.com

Hela Dellagi
hela.dellagi@outlook.com

December 28, 2024

Abstract

Catastrophic forgetting remains a major challenge when adapting large language models (LLMs) to new tasks or domains. Conventional fine-tuning often overwrites existing knowledge, causing performance degradation on original tasks. We introduce *Superposition in Transformers*, a novel architecture that leverages autoencoders to superimpose the hidden representations of a base model and a fine-tuned model within a shared parameter space. By using B-spline-based blending coefficients and autoencoders that adaptively reconstruct hidden states based on the input data distribution, our method effectively mitigates catastrophic forgetting and enables a new paradigm of "in-model" superposition. This approach preserves original model capabilities while allowing compact domain-specific expertise to be added, and it supports dynamic switching between model states during inference.

1 Introduction

Large language models (LLMs) such as GPT-3 [1] and GPT-4 [2] have shown remarkable performance on various language tasks. However, adapting these models to new tasks or domains often leads to **catastrophic forgetting**, where newly learned information overwrites older knowledge [3]. This challenge becomes critical in continual learning or domain adaptation settings, where preserving performance on original tasks is essential [4].

Mixture of Experts (MoE) [5] models mitigate catastrophic forgetting by delegating different distributions or tasks to distinct experts. However, MoE solutions typically require large parameter expansions and introduce complex gating. In contrast, we propose a more parameter-efficient method: we *blend* the hidden states of two models—a base model and a fine-tuned model—using trainable blending coefficients derived from B-splines. We then *reconstruct* these blended states via autoencoders, allowing each model's states to co-exist in a superposed fashion with minimal interference.

1.1 Contributions

- **Autoencoder-Based Superposition**: We demonstrate how autoencoders can reconstruct hidden states from either the base or fine-tuned model, effectively gating the representation space to prevent catastrophic forgetting.
- **Jointly Learned B-Spline Blending**: By training blending coefficients jointly with the autoencoders, we find a balance that preserves each model's capabilities while introducing minimal new parameters.
- **Parameter Efficiency and Practicality**: This approach retains the majority of each model's parameters in frozen form and only learns relatively small auxiliary modules, making it feasible for real-world use.

- **Future Directions**: We discuss possible expansions of this framework, such as merging specialized experts for chain-of-thought reasoning or leveraging token-level signals to dynamically switch hidden states within the same forward pass.

1.2 Motivation and Positioning

Most existing mixture-of-experts methods expand parameter count significantly by introducing distinct expert modules and gating networks [6, 7]. Parameter-efficient fine-tuning approaches such as Adapters [8] or LoRA [9] often rely on modifying or appending new weights to the base model. In contrast, our method produces a *single* merged set of parameters—via B-spline blending—while training autoencoders to reconstruct the original hidden states on demand. During training, both the base and fine-tuned model parameters are accessed in a frozen manner to generate hidden states, but in the end, only the new blended model parameters (plus the autoencoders) remain. This design merges the capabilities of two experts into one compact parameter space and mitigates catastrophic forgetting by preserving each model's knowledge in a superposed, reconstructable form.

2 Background and Related Work

2.1 Mixture of Experts

Mixture of Experts (MoE) dynamically allocates different experts to different inputs, improving capacity and specialization. While effective, MoE methods often increase parameter count and rely on gating networks [7]. Our method similarly aims to leverage multiple "expert" models but does so via a shared parameter space and lightweight autoencoder modules.

2.2 Parameter-Efficient Transfer Learning

Recent techniques like Adapters [8] and LoRA [9] reduce the cost of fine-tuning large models by injecting small trainable components. We differ in that we freeze both the base and fine-tuned models; instead of adding new layers directly on top of them, we *blend their hidden states* and *reconstruct* them adaptively. This approach can be viewed as orthogonal to Adapters or LoRA, potentially combining well with these methods.

2.3 Superposition and Polysemantic Neurons

In neural networks, *superposition* refers to efficiently encoding multiple features or tasks within the same parameter space [10]. Large LLMs often exhibit **polysemantic neurons**—units that respond to multiple unrelated concepts [11]. Our approach further promotes polysemanticity by forcing the network to encode the states of two distinct models in overlapping neurons, constrained by autoencoder bottlenecks.

2.4 Neural Network Compression and Model Merging

Prior methods for merging or compressing models [12, 13] generally aim to reduce memory overhead. By contrast, our framework focuses on *preserving both original models* in a single parameter space without overwriting. The added overhead is small (primarily the autoencoders and the blending coefficients) compared to training or storing two separate models.

3 Proposed Method

3.1 Overview

We merge a base model M_{base} and a fine-tuned model M_{fine} into a single "superposed" model M_{merged}. To do this, we:

1. **Blend Hidden States per Layer** using $\alpha(l)$, which we compute via **B-spline interpolation**.
2. **Insert Autoencoders** at selected layers to reconstruct the blended states either as $\mathbf{h}^{\text{base}}$ or $\mathbf{h}^{\text{fine}}$, based on input labels or domain cues.
3. **Train Jointly** the B-spline control points, layer biases, and autoencoder parameters so that each layer's $\alpha(l)$ best preserves domain-specific details when guided by the autoencoder reconstruction loss.

3.2 Blending Hidden States Using B-Splines

3.2.1 Motivation

Averaging or naively mixing model weights can degrade performance, as it does not adapt to which features are critical to each model. Instead, we *blend hidden states* directly, allowing each layer to remain mostly intact (frozen). The α-values are learned with a mechanism that encourages smooth transitions across layers.

3.2.2 Formulation

For layer l, let $\mathbf{h}_l^{\text{base}}$ and $\mathbf{h}_l^{\text{fine}}$ be the hidden states from the base and fine-tuned models, respectively. We define

$$\mathbf{h}_l = (1 - \alpha(l))\, \mathbf{h}_l^{\text{base}} + \alpha(l)\, \mathbf{h}_l^{\text{fine}},$$

where

$$\alpha(l) = \text{clamp}\Big(\sum_{i=1}^{N} c_i\, B_{i,k}(l) + b_l,\ 0,\ 1\Big).$$

- $\{c_i\}$ are trainable control points for a B-spline of degree k.
- b_l is a layer-specific bias term.
- The B-spline basis functions $B_{i,k}(\cdot)$ ensure smoothly varying $\alpha(l)$.

We freeze the weights of both M_{base} and M_{fine}, focusing on learning only $\{c_i\}$, $\{b_l\}$, and the autoencoders.

3.3 Merged Model Architecture

3.3.1 Merging Post-Training

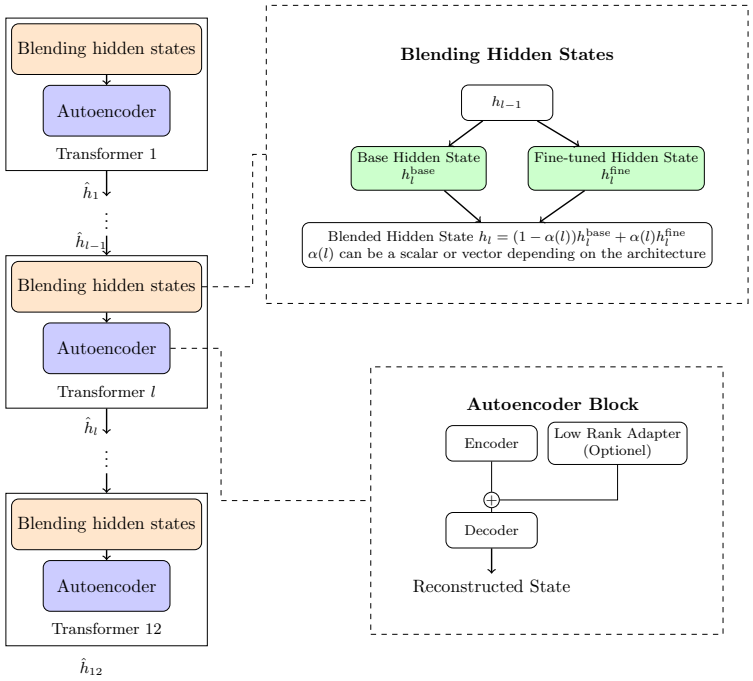
After training, one option is to produce a single model whose parameters reflect a final *hard merge* of the two original models:

$$\theta_l = (1 - \alpha(l))\, \theta_l^{\text{base}} + \alpha(l)\, \theta_l^{\text{fine}},$$

where θ_l^{base} and θ_l^{fine} are the layer parameters. This yields a standalone model with minimal overhead. Alternatively, one may keep the B-spline plus autoencoders as a gating system that dynamically adapts to inputs.

3.3.2 Forward Pass with Merged Model

During inference, the model computes hidden states for each layer via blended embeddings and self-attention (incorporating $\alpha(l)$) and optionally applies an autoencoder to refine $\mathbf{h}_l$, reconstructing the hidden states that best suit the input's domain. This structure preserves each model's domain knowledge while introducing minimal overhead.



3.4 Autoencoders for State Reconstruction

3.4.1 Architecture

At each selected layer l, an autoencoder compresses the blended hidden state $\mathbf{h}_l$ into a bottleneck and reconstructs it:

$$\mathbf{z}_l = \text{Encoder}(\mathbf{h}_l), \quad \hat{\mathbf{h}}_l = \text{Decoder}(\mathbf{z}_l).$$

We train it to match $\hat{\mathbf{h}}_l$ with either $\mathbf{h}_l^{\text{base}}$ or $\mathbf{h}_l^{\text{fine}}$. We *skip* embedding layers and the final layer norm to maintain alignment with the original model heads.

3.4.2 Minimizing Information Loss

Because α and the autoencoder parameters are trained jointly, α naturally adjusts so that the autoencoder can accurately reconstruct either $\mathbf{h}_l^{\text{base}}$ or $\mathbf{h}_l^{\text{fine}}$. This synergy ensures that if reconstruction error is high, $\alpha(l)$ shifts to a region that better preserves salient features from the respective domain.

3.4.3 Role of Autoencoders in Encouraging Polysemanticity

Each autoencoder's bottleneck forces the network to encode crucial details from both models within a limited dimension. This encourages **polysemantic** neurons, since the same hidden units may now carry signals relevant to both tasks. A narrower bottleneck heightens polysemantic pressure but can also degrade domain-specific detail; a wider bottleneck reduces polysemantic pressure but can maintain more domain specificity.

3.4.4 Extending to a 2D-alpha Model (Optional)

One can extend $\alpha(l)$ to a vector for per-dimension blending. In this scenario, local features (extracted by convolution layers) and global features (captured by a low-rank adapter) are combined in the autoencoder. While more expressive, this 2D-alpha approach also increases complexity and may demand careful tuning to avoid overfitting.

4 Training Procedure

4.1 Objectives

We optimize two core objectives:

- $\mathcal{L}_{\text{LM}}$: Standard language modeling loss, preserving fluency and coherence.
- $\mathcal{L}_{\text{Recon}}$: Reconstruction error between $\hat{\mathbf{h}}_l$ and the target hidden state $\mathbf{h}_l^{\text{base}}$ or $\mathbf{h}_l^{\text{fine}}$.

An additional alpha regularization term $\mathcal{L}_{\text{AlphaReg}}$ may be applied to keep α smooth or centered.

4.2 Loss Functions

We generally use:

$$\mathcal{L} \;=\; \lambda_{\text{LM}}\,\mathcal{L}_{\text{LM}} \;+\; \lambda_{\text{Recon}}\,\mathcal{L}_{\text{Recon}} \;+\; \lambda_{\text{Alpha}}\,\mathcal{L}_{\text{AlphaReg}},$$

where λ_{LM}, λ_{Recon}, and λ_{Alpha} balance the relative importance of each component. $\mathcal{L}_{\text{Recon}}$ is typically Mean Squared Error (MSE) or L2 distance over the hidden states.

4.3 Optimization

We freeze all parameters of M_{base} and M_{fine}. The trainable variables are the B-spline control points $\{c_i\}$ and biases $\{b_l\}$ as well as the autoencoder weights and biases per selected layer.
By iterating through mini-batches, we compute hidden states from both models (frozen), blend them via $\alpha(l)$ to produce $\mathbf{h}_l$ and use an autoencoder to reconstruct $\mathbf{h}_l$ into either $\mathbf{h}_l^{\text{base}}$ or $\mathbf{h}_l^{\text{fine}}$. We then backpropagate the reconstruction error to update α and the autoencoder parameters. The final forward pass for language modeling also contributes to $\mathcal{L}_{\text{LM}}$, ensuring that fluency is maintained.

4.4 Role of Labels

Labels indicating whether an input belongs to the base or fine-tuned domain can guide which hidden state the autoencoder targets. While optional in principle, these labels can significantly speed up convergence and improve accuracy of reconstruction for each domain.

5 Experiments and Results

In this section, we evaluate the proposed merged model, by integrating a base GPT-2 model and a fine-tuned GPT-2 model trained on French text. We focus on demonstrating how the autoencoders enable the superposition of transformer blocks from different fine-tunes, allowing the merged model to effectively combine representations from both models. Our analysis includes performance metrics, hidden state reconstruction and the emergence of polysemantic neurons.

5.1 Experimental Setup

We compare the following models:

- **Base Model**: GPT-2 trained on English data.
- **Fine-Tuned Model**: GPT-2 fine-tuned on a French corpus.
- **Merged Model**: Combines the base and fine-tuned models using layer-wise blending with learned α values and incorporates autoencoders to enable superposition.

Datasets English and French evaluation datasets:

- **English Dataset (DS1)**: A subset of the GPT-2 training data (6,000 samples repeated with range(3) in the 1D experiment and repeated with range(2) in the 2D experiment).
- **French Dataset (DS2)**: 18,000 French Wikipedia articles for the 1D model and 12,000 French Wikipedia articles for the 2D model.

10% of the combined samples are reserved for validation.

In the case of the 1D-alpha GPT2 model the number of parameters per autoencoder is:
$4 \times$ dimension of the hidden state $\times$ bottleneck size + dimension of the hidden state + bottleneck size
The bottleneck size used in the 1D-alpha model experiment is 576, resulting in approximately 1.7 million parameters per autoencoder.

5.2 Results of the 1D model

5.2.1 Overall Performance

To empirically demonstrate the advantages of our *Superposition in Transformer* technique, we conducted experiments comparing Superposition Merging with linear interpolation and task arithmetic techniques. The perplexity of the autoencoder-merged model (**M-PPL=47.01**) was markedly lower than that of the linearly interpolated model (**I-PPL=60.29**) and the task arithmetic model (**I-PPL=61.30**) during the last epoch of training, suggesting a higher confidence in predicting the next token. The merged model initially showed higher perplexity than both task arithmetic and linear interpolation models but then achieved lower values as training progresses (figure 1).
Similarly, the next-token prediction accuracy was also improved with Superposition Merging (**M-Acc=0.3270**) compared to linear interpolation (**I-Acc=0.3039**) and task arithmetic (**I-Acc=0.2957**).

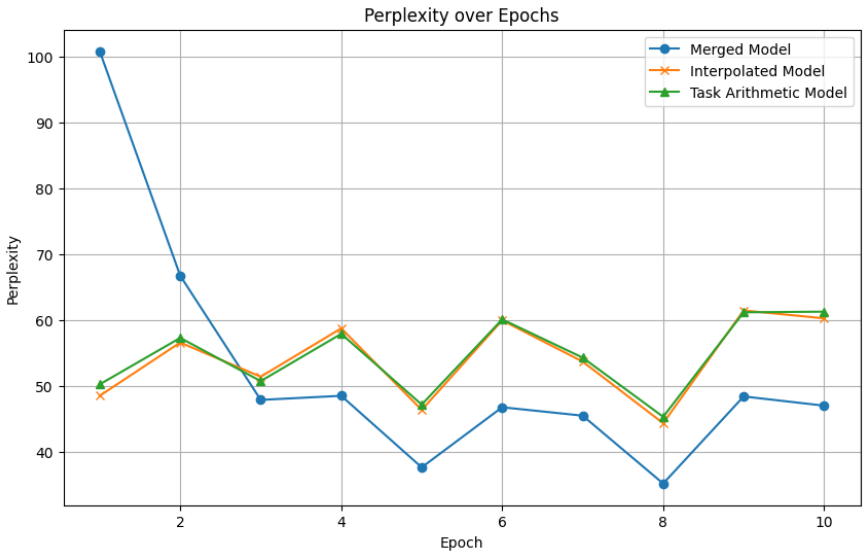


Figure 1: Perplexity evolution across epochs for different merging methods.

Additionnaly, we examined the Jensen-Shannon Divergence JSD values over different epochs during training. The JSD drops sharply from **M-JSD=82.9** to **M-JSD=45.5** in the first two epochs. This drastic reduction indicates that the merged model's output distribution quickly diverges from the initial average of the base and fine-tuned models. After this initial drop, the JSD values fluctuated between approximately

M-JSD=36.5 and **M-JSD=44.2**, suggesting that the merged model settles into a relatively stable output distribution that is distinct from the simple average.

These quantitative results strongly suggest that Superposition Merging, facilitated by the learning capabilities of autoencoders, offers a more robust and effective method for combining the strengths of independently trained models. Unlike traditional methods that can lead to a homogenization of expertise, our approach shows promising potential for achieving a genuine superposition, where the merged model can effectively leverage the unique skills of each expert model within their respective domains. This opens up exciting possibilities for creating more versatile and powerful AI systems by intelligently combining specialized knowledge.

5.3 Results of the 2D model

5.3.1 Visualization of Hidden States in Layer 4

To evaluate how the autoencoders enable the adaptive blending of representations in the 2D-alpha model, a t-SNE analysis is conducted on the hidden states from layer 4. Similar to the 1D-alpha model analysis, Figure 5 effectively illustrates the model's capability to align reconstructed hidden states with the appropriate model based on input language. Specifically, for English inputs, the reconstructed states from the merged model are closely aligned with the base model, indicated by the use of "○" markers in the graph. Conversely, for French inputs, these states align more closely with the fine-tuned model, which is depicted using "x" markers. This distinction visually underscores the model's adaptive capacity to toggle between the base and fine-tuned model states depending on the language context

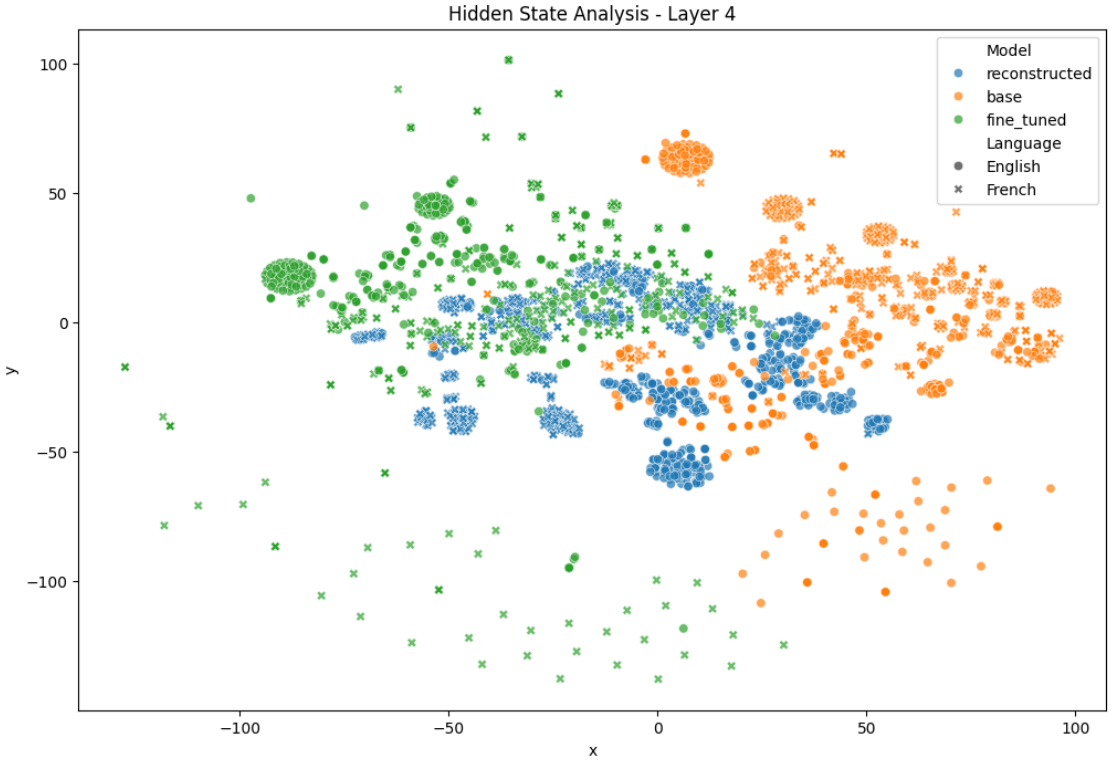


Figure 2: t-SNE visualization of layer 4 hidden states from the merged model and expert models for English and French inputs (2D-alpha).

5.3.2 Neuron Utilization and Polysemantic Neurons

The merged model maintains efficient neuron utilization and shows an increase in polysemantic neurons, as shown in Table 6:

Model	Sparsity (%)	Polysemantic Neurons (%)
Base Model	0.0166	1.8229
Fine-Tuned Model	0.0154	2.0833
Merged Model	0.0179	3.3854

Table 1: Sparsity and proportion of polysemantic neurons in layer 4 of the models.

The 2D-alpha model demonstrates enhanced neuron utilization. The increase in sparsity is minimal and potentially arbitrary due to the model's small size.
The percentage of polysemantic neurons is computed using a threshold value of 0.05. This threshold delineates neurons based on the normalized difference in their average activations across different language contexts (e.g., English vs. French). Neurons with a normalized difference below this threshold are considered polysemantic because they react similarly across linguistic contexts, suggesting a shared semantic load. However, using such a fixed threshold might not reliably capture the full spectrum of neuron functionality or their sensitivity to different contexts. Given the potential limitations of using a fixed threshold for identifying polysemantic neurons, the comparative analysis of neuron diversity and activation employs a more dynamic approach.

5.3.3 Comparative Analysis of Neuron Diversity and Activation

To gain deeper insights into how the 2D-alpha merged model utilizes neurons compared to the base and fine-tuned models, we conducted an analysis of neuron diversity and activation across layers. Instead of defining polysemicity with a single threshold, this analysis evaluates the diversity of neuron responses through a MiniBatchKMeans clustering algorithm,. The diversity is quantified using entropy measures derived from the distribution of neurons across clusters formed based on their activation patterns. This method provides a deeper insight into how neurons respond to different linguistic inputs, reflecting a broader range of neuron functionality than can be detected by a simple threshold-based method.
Note: In the 2D-alpha model, the autoencoders were used starting from layer 4 until layer 10. The graphs below clearly illustrate the contrast between layer 3 and the subsequent layers 4, 5, and 6.

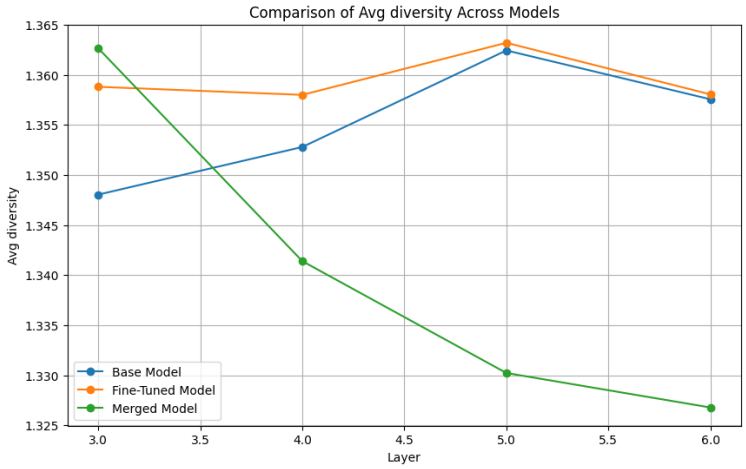


Figure 3: Comparison of average neuron diversity across layers for the base, fine-tuned, and merged models.

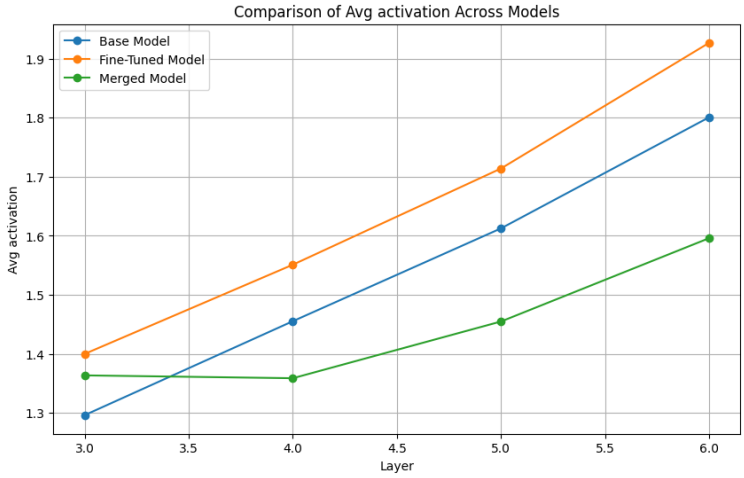


Figure 4: Comparison of mean neuron activation across layers for the base, fine-tuned, and merged models.

- The reduction in diversity metrics shows that neurons encode overlapping or generalized features. While layers without (e.g. layer 3) autoencoders retain higher diversity due to the presence of specialized neurons, the autoencoder forces neurons to unify and encode features relevant across multiple tasks or contexts. This suggests a shift from task-specificity to multi-purpose functionality—a hallmark of polysemantic neurons.
- The lower average activation magnitudes observed in pre-autoencoder layers reflect a shift towards encoding distributed, low-magnitude representations. This subtle encoding strategy ensures that the autoencoder can reconstruct activations effectively while maintaining compactness. Distributed, low-magnitude activations are more likely to encode polysemantic features, as they provide flexibility and reuse across tasks.

6 Future Work

- **Tuning Bottleneck Hyperparameters**: Experimenting with different bottleneck sizes can help balance the trade-off between generalization and task-specificity.
- **Analyzing Cross-Layer Interactions**: Investigate how the presence of an autoencoder in one layer influences subsequent layers. Does the polysemanticity induced in one block propagate through the network?
- **Enable Language Switching Within a Single Context**: A promising direction is enabling the model to alternate between languages within a single context, such as generating output that seamlessly switches between English and French while maintaining coherence and fluency. This could be particularly valuable for tasks requiring multilingual communication, such as translation or conversational agents operating in diverse linguistic environments. Special tokens or contextual cues could signal the model to adjust its representations dynamically, ensuring that the transitions between languages preserve the original intent and meaning of the sentence.
- **Merging Specialized Experts**: We plan to explore merging a symbolic reasoning expert with a domain knowledge expert. This would enable the model to switch between symbolic processing and contextual understanding dynamically. This idea is analogous to the language switching above, as it allows seamless transitions between different modes of cognition while preserving the overarching context.
- **Single Pass Generation**: Similar to the language switching experiment, special tokens or cues could be used to signal the model to switch states within a single generation sequence.

- **Example Scenario**:
  - *Input*: "Solve for x: $2x + 3 = 7$. Then, explain the significance of the solution in real-world applications."
  - *Expected Behavior*:
    - **First Part (Symbolic Reasoning)**: The model uses the symbolic reasoning expert to solve the equation.
    - **Second Part (Domain Explanation)**: The model switches to the domain knowledge expert to provide an explanation.

7 Conclusion

The proposed method represents a significant advancement in enabling large language models (LLMs) to integrate knowledge from multiple domains or tasks without sacrificing foundational capabilities or requiring extensive parameter growth. By leveraging autoencoders and blending mechanisms, the approach facilitates efficient knowledge integration making it a practical and resource-efficient enhancement to LLMs.
Experimental results highlight improvements in perplexity reduction, reconstruction accuracy, and hidden state alignment with base and fine-tuned models. Techniques like t-SNE visualizations further validate the model's ability to adapt representations dynamically. This adaptability enables seamless processing of diverse inputs and enhances multi-domain integration.
Beyond merging two models, this method promises broader applications, such as multilingual processing, seamless language switching, and integrating symbolic reasoning with domain knowledge.
Looking ahead, this approach paves the way for the development of models capable of integrating diverse skills and knowledge domains, representing a step toward more flexible, powerful, and general-purpose AI systems. The potential applications are vast and exciting.

References

[1] Tom B. Brown and al., "Language Models are Few-Shot Learners," *arXiv preprint arXiv:2005.14165*, 2020.

[2] OpenAI., "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology of Learning and Motivation*, vol. 24, pp. 109–165, 1989.

[4] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[5] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[6] 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 *International Conference on Learning Representations*, 2017.

[7] Longfei Yun, Yonghao Zhuang, Yao Fu, Eric P Xing, Hao Zhang. Toward Inference-optimal Mixture-of-Expert Large Language Models *arXiv preprint arXiv:2404.02852*, 2024.

[8] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[9] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[10] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer and al. Toy Models of Superposition. *arXiv preprint arXiv:2209.10652*, 2022.

[11] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom In: An Introduction to Circuits. *Distill*, 2020.

[12] Tao Li, Lei Deng, Sheng Lin, Meng Li, Dong Huang, and Yuan Xie. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on Machine Learning*, 2020.

[13] William Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *International Conference on Learning Representations*, 2022.

A Conventions

In this paper, we refer to the original, unmodified model as the base model, denoted by M_{base} and to the fine-tuned version, referred to as M_{fine}. We have implemented two distinct architectures: the first serves as a simplified implementation (in terms of number of parameters) to demonstrate the aforementioned properties.This model refered to as the "1D-alpha model" due to its use of scalar alpha values for layer-wise adjustments. The second delves deeper into exploring the impact of the autoencoder on the polysemantic nature of neurons within a transformer block and is refered to as the "2D-alpha model" due to its utilization of vector-based alpha values for each layer. Collectively, any form of these models will be referred to as the merged model M_{merged}.
L denotes the number of transformer blocks within the model, and D is the hidden size of each transformer block.

B Training Algorithms

B.1 Loss Functions

The losses involved in the different architectures combines:

Reconstruction Loss $\mathcal{L}_{\text{Recon}}$ This loss measures how well the autoencoders reconstruct the blended hidden states. It uses both Mean Squared Error (MSE) and the L2 distance between the reconstructed hidden states and the target hidden states.

$$\mathcal{L}_{\text{Recon}} = \sum_{l=1}^{L} \left[\lambda_{\text{MSE}} \mathbb{E} \left[\left\| \hat{\mathbf{h}}_l - \mathbf{h}_l^{\text{target}} \right\|_2^2 \right] + \lambda_{\text{L2}} \left\| \hat{\mathbf{h}}_l - \mathbf{h}_l^{\text{target}} \right\|_2 \right] \tag{1}$$

L represents the number of layers in the transformer model.

Language Modeling Loss $\mathcal{L}_{\text{LM}}$ This loss measures the standard language modeling objective using cross-entropy between the predicted logits and the next token in the sequence.

$$\mathcal{L}_{\text{LM}} = \mathbb{E} \left[-\sum_{t} \log P_\theta(w_t | w_{<t}) \right] \tag{2}$$

Alpha Regularization Loss $\mathcal{L}_{\text{AlphaReg}}$ The alpha regularization loss encourages the control points of the B-spline-based blending coefficients to adhere to desirable properties. It ensures smooth and interpretable blending, avoiding overfitting to noisy representations. It includes three components:

- **Smoothness Loss**: Ensures that adjacent control points are close to each other, minimizing abrupt changes in blending behavior.

$$\mathcal{L}_{\text{Smoothness}} = \sum_{i=1}^{N-1} \|\mathbf{c}_i - \mathbf{c}_{i-1}\|_2^2 \tag{3}$$

  N represents the number of control points in the B-spline interpolation.

- **Centrality Loss**: Penalizes deviations of the control points from a central value (e.g., 0).

$$\mathcal{L}_{\text{Centrality}} = \sum_{i=1}^{N} \|\mathbf{c}_i\|_2^2 \tag{4}$$

- **Mean Bias Loss**: Penalizes deviations of the mean layer-wise bias from zero to encourage balanced adjustments across layers.

$$\mathcal{L}_{\text{MeanBias}} = \mu_b^2 \tag{5}$$

  where μ_b^2 is the mean of the layer-wise bias.

- **Variance Bias Loss**: Encourages the variance of the layer-wise bias to match a desired target variance σ_{target}^2.

$$\mathcal{L}_{\text{VarianceBias}} = (\sigma_b^2 - \sigma_{target}^2)^2 \tag{6}$$

  where σ_b^2 is the variance of the layer-wise bias.

The total alpha regularization loss is:

$$\mathcal{L}_{AlphaReg} = \lambda_{Smoothness}\mathcal{L}_{Smoothness} + \lambda_{Centrality}\mathcal{L}_{Centrality} + \lambda_{MeanBias}\mathcal{L}_{MeanBias} + \lambda_{VarianceBias}\mathcal{L}_{VarianceBias} \tag{7}$$

The Total Losses The total 1D loss combines the language modeling loss and the reconstruction loss. Each component is weighted to balance their contributions during training.

$$\mathcal{L}_{\text{1D}} = \lambda_{\text{Recon}}\mathcal{L}_{\text{Recon}} + \lambda_{\text{LM}}\mathcal{L}_{\text{LM}} \tag{8}$$

The total 2D loss combines all the aforementionned losses weighted to balance their contributions during training.

$$\mathcal{L}_{\text{2D}} = \lambda_{\text{Recon}}\mathcal{L}_{\text{Recon}} + \lambda_{\text{LM}}\mathcal{L}_{\text{LM}} + \lambda_{\text{AlphaReg}}\mathcal{L}_{\text{AlphaReg}} \tag{9}$$

B.2 1D-alpha Model Training algorithm

Algorithm 1 Merging Two LLMs with B-spline Weight Blending and Autoencoders

Require: Base model parameters $\{\theta_l^{\text{base}}\}_{l=1}^{L}$, fine-tuned model parameters $\{\theta_l^{\text{fine}}\}_{l=1}^{L}$, number of layers L, number of control points N, B-spline degree k, control point parameters $\{c_i\}_{i=1}^{N}$, autoencoder parameters $\{\phi_l\}_{l=1}^{L}$

Ensure: Merged model parameters $\{\theta_l\}_{l=1}^{L}$, trained autoencoders $\{\text{AE}_l\}_{l=1}^{L}$

Compute blending coefficients $\alpha(l)$ for each layer l using B-spline interpolation:
for $l = 1$ to L **do**
  $\alpha(l) \leftarrow \text{clamp}\left(\sum_{i=1}^{N} c_i B_{i,k}(l) + b_l,\ 0,\ 1\right)$
end for
Merge model parameters using the blending coefficients:
for $l = 1$ to L **do**
  $\theta_l \leftarrow (1 - \alpha(l))\theta_l^{\text{base}} + \alpha(l)\theta_l^{\text{fine}}$
end for
Initialize autoencoders $\{\text{AE}_l\}_{l=1}^{L}$ with parameters $\{\phi_l\}_{l=1}^{L}$
Training Phase:
while not converged **do**
  Sample a minibatch of inputs $\mathbf{x}$ and targets $\mathbf{y}$
  Forward pass through the merged model:
  $\mathbf{h}_0 \leftarrow \text{Embed}(\mathbf{x})$
  for $l = 1$ to L **do**
    $\mathbf{h}_l \leftarrow \text{TransformerLayer}(\hat{\mathbf{h}}_{l-1}; \theta_l)$
    Autoencoder processing at layer l:
      Encoder:
      $\mathbf{z}_l \leftarrow \sigma_{\text{enc}}(f_{enc}^{1D}(\mathbf{h}_l))$
      Decoders:
      $\hat{\mathbf{h}}_l \leftarrow f_{dec}^{1D}(\mathbf{z}_l)$
  end for
  Compute output logits:
  $\text{logits} \leftarrow \text{LMHead}(\hat{\mathbf{h}}_L)$
  Compute losses:
    $\mathcal{L}_{\text{LM}} \leftarrow \text{CrossEntropyLoss}(\text{logits}, \mathbf{y})$
    $\mathcal{L}_{\text{Recon}} \leftarrow \sum_{l=1}^{L} \left[\lambda_{\text{MSE}} \mathbb{E}\left[\left\|\hat{\mathbf{h}}_l - \mathbf{h}_l^{\text{target}}\right\|_2^2\right] + \lambda_{\text{L2}} \left\|\hat{\mathbf{h}}_l - \mathbf{h}_l^{\text{target}}\right\|_2\right]$
  Compute total loss:
  $\mathcal{L}_{\text{1D}} \leftarrow \lambda_{\text{LM}}\mathcal{L}_{\text{LM}} + \lambda_{\text{Recon}}\mathcal{L}_{\text{Recon}}$
  Backpropagate and update control points $\{c_i\}$ and autoencoder parameters $\{\phi_l\}$
end while
Inference Phase:
Use merged model parameters $\{\theta_l\}$ and autoencoders $\{\text{AE}_l\}$ for inference

B.3 2D-alpha Model Training algorithm

Algorithm 2 Merging Two LLMs with 2D B-spline Weight Blending and Dual-Pathway Autoencoders

Require: Base model parameters $\{\theta_l^{\text{base}}\}_{l=1}^{L}$, fine-tuned model parameters $\{\theta_l^{\text{fine}}\}_{l=1}^{L}$, number of layers L, number of control points N, B-spline degree k, control point parameters $\{c_i\}_{i=1}^{N}$, autoencoder parameters $\{\phi_l\}_{l=1}^{L}$

Ensure: Merged model parameters $\{\theta_l\}_{l=1}^{L}$, trained autoencoders $\{\text{AE}_l\}_{l=1}^{L}$

Compute dimension-wise blending coefficients $\alpha(l)$ for each layer l and vector V_d:
for $l = 1$ to L **do**
  $\alpha(l) \leftarrow \text{clamp}\left(\sum_{i=1}^{N} c_i \odot B_{i,k}(l) + b_l,\ 0,\ 1\right)$
end for
Merge model parameters using the blending coefficients:
for $l = 1$ to L **do**
  $\theta_l = (\mathbb{1} - \alpha(l)) \odot \theta_l^{\text{base}} + \alpha(l) \odot \theta_l^{\text{fine}}$
end for
Initialize autoencoders $\{\text{AE}_l\}_{l=1}^{L}$ with parameters $\{\phi_l\}_{l=1}^{L}$
Training Phase:
while not converged **do**
  Sample a minibatch of inputs $\mathbf{x}$ and targets $\mathbf{y}$
  Forward pass through the merged model:
  $\mathbf{h}_0 \leftarrow \text{Embed}(\mathbf{x})$
  for $l = 1$ to L **do**
    $\mathbf{h}_l \leftarrow \text{TransformerLayer}(\hat{\mathbf{h}}_{l-1}; \theta_l)$
    Autoencoder processing at layer l:
      Local Pathway (Convolutions):
      $\mathbf{z}_l^{\text{local}} \leftarrow \sigma_{\text{local}}\left(f_{local}^{2D}(\mathbf{h}_l)\right)$
      Global Pathway (Residual Adapter):
      $\mathbf{z}_l^{\text{global}} \leftarrow f_{global}^{2D}(\mathbf{h}_l)$
      Combine Bottleneck Representations:
      $\mathbf{z}_l \leftarrow \text{Concat}(\mathbf{z}_l^{\text{local}}, \mathbf{z}_l^{\text{global}})$
      Decoder Reconstruction:
      $\hat{\mathbf{h}}_l \leftarrow f_{dec}^{2D}(\mathbf{z}_l)$
  end for
  Compute output logits:
  $\text{logits} \leftarrow \text{LMHead}(\hat{\mathbf{h}}_L)$
  Compute losses:
    $\mathcal{L}_{\text{LM}} \leftarrow \text{CrossEntropyLoss}(\text{logits}, \mathbf{y})$
    $\mathcal{L}_{\text{Recon}} \leftarrow \sum_{l=1}^{L} \left[\lambda_{\text{MSE}} \mathbb{E}\left[\left\|\hat{\mathbf{h}}_l - \mathbf{h}_l^{\text{target}}\right\|_2^2\right] + \lambda_{\text{L2}} \left\|\hat{\mathbf{h}}_l - \mathbf{h}_l^{\text{target}}\right\|_2\right]$
    $\mathcal{L}_{\text{AlphaReg}} \leftarrow \lambda_{\text{Smoothness}}\mathcal{L}_{\text{Smoothness}} + \lambda_{\text{Centrality}}\mathcal{L}_{\text{Centrality}} + \lambda_{\text{MeanBias}}\mathcal{L}_{\text{MeanBias}} + \lambda_{\text{VarianceBias}}\mathcal{L}_{\text{VarianceBias}}$
  Compute total loss:
  $\mathcal{L}_{\text{2D}} \leftarrow \lambda_{\text{LM}}\mathcal{L}_{\text{LM}} + \lambda_{\text{Recon}}\mathcal{L}_{\text{Recon}} + \lambda_{\text{AlphaReg}}\mathcal{L}_{\text{AlphaReg}}$
  Backpropagate and update control points $\{c_i\}$ and autoencoder parameters $\{\phi_l\}$
end while
Inference Phase:
Use merged model parameters $\{\theta_l\}$ and autoencoders $\{\text{AE}_l\}$ for inference

C Additional results of the 1D-alpha Model experiment

Tables 1 and 2 compare the perplexity of the models using French and English inputs :

Model	Language	Perplexity
Base Model	English	28.88
	French	132.02
Fine-Tuned Model	English	54.42
	French	29.57
Merged Model	English	43.14
	French	33.20

Table 2: Perplexity of the models on the evaluation datasets

The perplexity results show distinct performance characteristics for the models. For French inputs, the fine-tuned model has a significantly lower perplexity (29.57) compared to the base model (132.02), indicating its suitability for French. Conversely, the base model performs best on English inputs with a lower perplexity (28.88), suggesting it was primarily optimized for English. The merged model demonstrates comparable perplexity to the base model in English (43.14 vs. 28.88) and is close to the fine-tuned model in French (33.20 vs. 29.57), indicating its effectiveness in bridging the performance across both languages.

C.1 Visualization of Hidden States in Layer 4

To further illustrate how the autoencoders enable the superposition of representations, we perform a t-SNE visualization of the hidden states from layer 4. Layer 4 is chosen as an early intermediate feature specialized block.

Figure 5: t-SNE visualization of layer 4 hidden states from the merged model and expert models for French inputs.

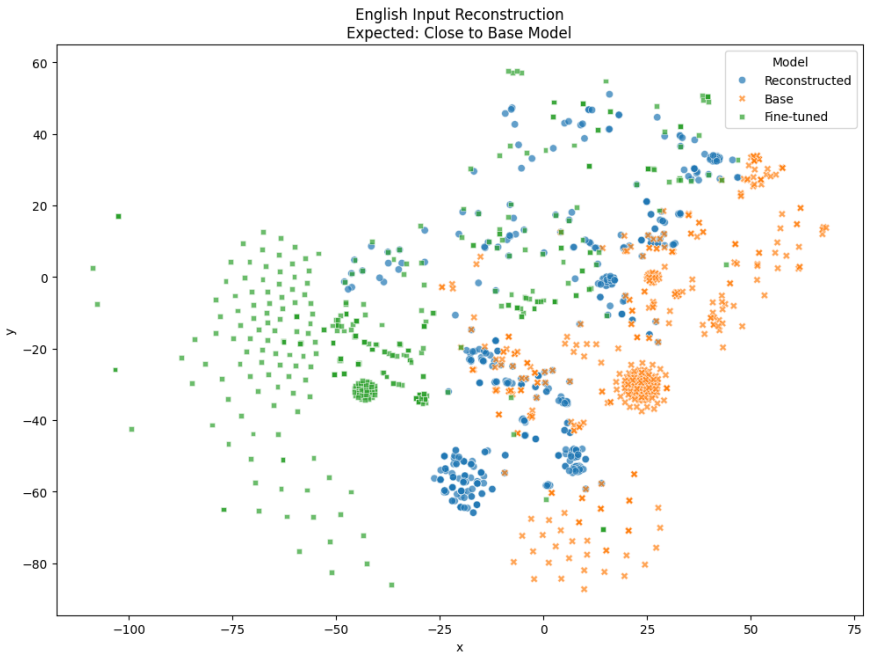


Figure 6: t-SNE visualization of layer 4 hidden states from the merged model and expert models for English inputs.

Analysis The t-SNE projection in Figure 2 provides a visual representation of how the merged model's layer 4 hidden states are reconstructed based on the input data. Specifically:

- **English Inputs**: The reconstructed hidden states of the merged model (blue points) cluster closely with the hidden states of the base model (green points). This indicates that for English inputs, the autoencoders adjust the blended hidden states to align with the base model's representations.
- **French Inputs**: The reconstructed hidden states of the merged model (red points) cluster near the hidden states of the fine-tuned model (orange points). This shows that for French inputs, the autoencoders reconstruct the hidden states to resemble those of the fine-tuned model.

This behavior demonstrates that the autoencoders effectively condition the merged model's internal representations based on the input language. By reconstructing the hidden states to match the appropriate expert model, the merged model can adaptively process different languages within a unified framework.

Implications for Superposition The t-SNE visualization highlights the role of autoencoders in enabling superposition:

- **Adaptive Representation**: The merged model produces hidden states that are contextually aligned with the relevant expert model, showcasing its ability to adapt representations based on input data.
- **Efficient Parameter Usage**: By superimposing the representations within the same parameters, the model avoids redundancy and efficiently utilizes its capacity.

C.2 Evaluation of Hidden State Trajectories

To evaluate the behavior of the 1D Merged Model, we analyzed the trajectories of hidden states across selected layers in both English and French language samples. This analysis helps to visually understand how the merged model acts given different input. Principal Component Analysis (PCA) was used to project the high-dimensional hidden states into a 2D space, preserving the primary variance for visualization. The hidden states were extracted for layers 3, 4, 5, 6, and 7 from the base, fine-tuned, and merged models so

the markers in figure (3) indicated the start (layer 3) and end (layer 7) points, and annotations denoted the layers.

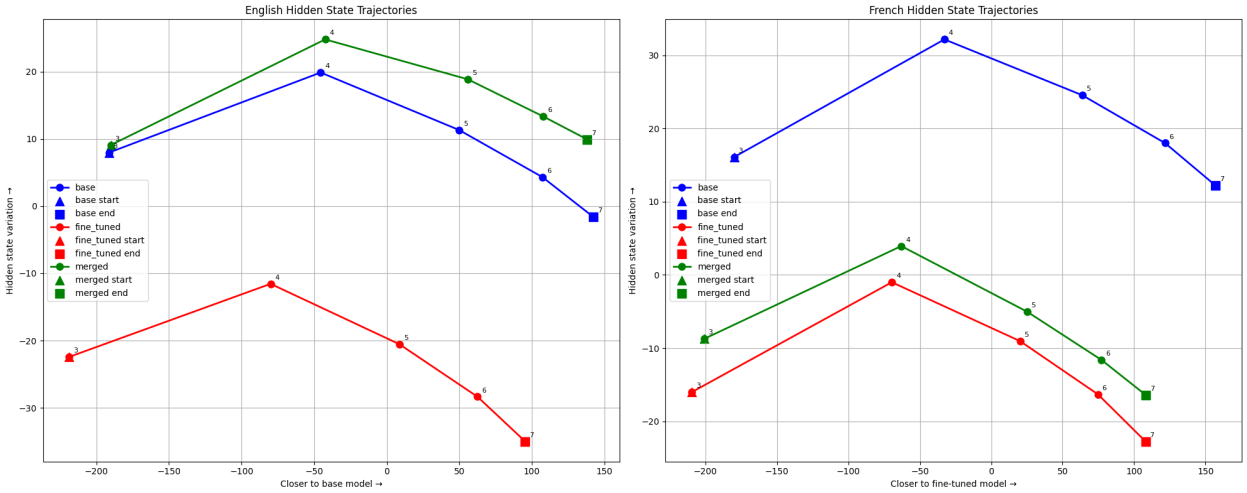


Figure 7: PCA visualization of hidden state trajectories from the base, fine-tuned, and merged models across layers 3 to 7 for English and French inputs.