ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Mechanistic Interpretability

INVESTIGATING MECHANISTIC INTERPRETABILITY IN LARGE LANGUAGE MODELS THROUGH MODEL STITCHING ACROSS ARCHITECTURES

CANDIDATE
Luca Tedeschini

SUPERVISOR
Prof. Paolo Torroni

CO-SUPERVISORS
Andrea Capitani
Dott. Federico Ruggeri

Academic year 2025-2026
Session 5st

Contents

List of Figures

List of Tables

Abstract

Mechanistic interpretability aims to elucidate the internal reasoning of Large Language Models (LLMs) by reverse-engineering their latent representations. It is a very broad field of research, that spans across many different hypothesis and research questions. Since it is impossible to provide a complete logical and mathematical explanation of how an LLM reasons and operates, research instead focuses on testing and validating small hypotheses that, collectively, help build a broader understanding. For example, the work of Chen et al. [4] has demonstrated the feasibility of ”model stitching” within the same model family. Their work aims to prove that an affine mapping exists between residuals stream of language models, and that this affine mapping can be use in practical ways, such as transferring sparse autoencoders' (SAEs) weights. This work extends their investigation to inter-family stitching, addressing the challenge of bridging models with distinct architectures and tokenizers. In the context of interpretability, model stitching is a tool that helps us understand and measure how two models differ. It does so through a “reverse engineering” approach: once two models are stitched together, we can analyze how the stitching operates, potentially revealing new insights about the individual models themselves. Focusing on open-weights models such as Llama-3, Gemma-2, and Qwen-2.5, this thesis explores the alignment of latent spaces using affine transformations and sparse autoencoders. The methodology investigates the transferability of hidden states across different families, tackling the vocabulary mismatch problem through dataset alignment and direct latent mapping. Experimental results on standard benchmarks, including HellaSwag and MMLU, indicate that functional hybrid models can be constructed with performance degradation comparable to intra-family baselines. In particular, the Llama-3 to Qwen-2.5 configuration demonstrates significant semantic coherence. Additionally, the research highlights the ”Single Tokenizer” phenomenon: when two models are stitched together, it is possible to use the tokenizer of the first model to decode the outputs of the second model B (without modifying the second model weights) and still obtain a functioning

system. This suggests that high-level semantic alignment between the models can allow one of them to operate effectively within an alien token space. These findings suggest that, despite architectural differences, LLMs converge towards linearly mappable internal representations, supporting the broader hypothesis of a shared semantic space.

Chapter 1

Introduction

Mechanistic interpretability seeks to reverse engineer neural networks, similar to how one might reverse engineer a compiled binary computer program. After all, neural network parameters are in some sense a binary computer program which runs on one of the exotic virtual machines we call a neural network architecture ~ Chris Olah. [12]

Mechanistic interpretability, as a field, is quite recent. Until now, development and focus have been centered on creating new architectures, often overlooking the fact that most current Artificial Intelligence (AI) models are *black boxes*[1]. Dario Amodei, CEO of Anthropic, argued in a blog post [1] that interpretability and technological progress should advance in parallel. He posited that only by fully understanding how these systems work can we truly improve them and overcome their limitations.
Moreover, the recent proliferation of Large Language Models (LLMs) as chatbots, such as ChatGPT [14], Gemini [7], and Perplexity [16], has democratized the usage of such models. Their rapid technological advancement in such a short period, including multimodal support and media generation, has created many social dilemmas that still need to be answered: is it acceptable to use them in serious context, for example in courts or in medical fields? Can we trust these systems in mission critical environment? Are they safe, or can be made safe, for underage users?

[1] A black box AI is an AI system whose internal workings are a mystery to its users. Users can see the system's inputs and outputs, but they can't see what happens within the AI tool to produce those outputs. [10]

For example, the Italian Minister of Education, Giuseppe Valditara, stated in an interview [17] that he is interested in the opportunities such models offer in an educational context, while emphasizing the importance of human interaction in the classroom. OpenAI has also highlighted the health-related capabilities of its newest model, GPT-5 [15], suggesting it could be useful for addressing users' concerns about their health. Many people have started using LLM-based chatbots instead of Google (or other search engines) [11]. But how can we be confident in delegating the responsibility of verifying sources and claims to a model that often hallucinates providing false or plainly wrong information [9]?
The term "*hallucination*", in the context of LLMs, refers to instances in which a model generates false or illogical information while presenting it in a very confident manner. This can be potentially dangerous, as it may lead to harmful consequences if the correctness of the model's output is taken for granted.
For this reason, the field of interpretability is critically important: understanding how these models "*reason*" could help us prevent, and even correct, hallucinations at their root cause or just before the next token is generated. This would allow them to be used in more critical contexts, such as education or even mental health. As of today, however, this goal is still out of reach.

1.1 Contributions and Outlines

This work tackles mechanistic interpretability from a high-level perspective, seeking to understand if different models, based on slightly different architectures and trained on different data, share a common *knowledge space*. It also explores how we can transfer the "*thinking process*" of one LLM to another by stitching the two models together. These hypotheses are based on the *semantic hub hypothesis* [24], which posits that the capabilities learned by a language model emerge from a shared representation space across heterogeneous data types. If proven, the semantic hub hypothesis would be a significant step forward for interpretability, as it would simplify the entire process. Instead of studying each LLM individually, research efforts could be focused on studying the semantic hub, and the findings would apply to all existing Transformer-based LLMs, since they would all share this common

knowledge space.
This work takes this hypothesis and applies it to model stitching between models from different families, with the goal of achieving a functional hybrid model. The objective is not to create a better model or to optimize computation, but to demonstrate that such an operation is possible.
The following sections detail how the work was conducted. An analysis of related work (2) situates this thesis within the current state of the art in mechanistic interpretability. Chapter 3 then introduces the fundamental building blocks upon which the main experiments are based. Finally, Chapters 4 and 5 present the hybrid model architecture and evaluate it across different benchmarks. Chapter 6 concludes the thesis by outlining possible future work and summarizing the main findings.

Chapter 2

Related Work

This work is heavily inspired by two strands in current literature: "*model stitching*" and "*activation steering*".

2.1 Model Stitching

Model stitching is the task of aligning the latent space of two different models in order to create a bridge capable of translating the latent space of a giver model to the latent space of a receiving model. The goal of this task is to evaluate how functionally similar the representations learned by independently trained neural networks are. In the context of large language model, the task can be rephrased as how similarly two independently trained large language models represent natural language. This task is directly linked to the semantic hub hypothesis: if the semantic hub hypothesis were demonstrated to be true, then model stitching would be almost trivial, because learned representations of different models of natural language would converge to a universal representation; it would be easy to translate one representation to another. Unfortunately, the success of model stitching does not imply that the semantic hub hypothesis is true, but it is indeed a step forward towards that direction.
The work of Chen et al. [4] tackles model stitching by posing strong assumptions: within the same model family, i.e, models trained on the same training data by the same team using almost identical architecture, model stitching is possible, and it is achievable using an affine transformation. They test the degree of success of hybrid models by measuring the degradation in performance

in the next token prediction task, which is the standard training objective for large language models. Most notably, they observe that a hybrid model created by stitching a small model (A) to a bigger model (B), and then stitching the same bigger B model back to model A (the stream of information gets processed by A, then B and then again by A) produces a negligible degradation in performance, meaning that the translation process is practically lossless. To achieve a functional stitching, they assume that the translation is bidirectional (i.e. from a layer in model B it is possible to reconstruct the output of a layer in model A), hence the loss signal in their training is produced by the reconstruction error of two transformation matrices: one that transforms states from A to B, and another that transforms states from B to A. Once the training is finished, they discard the second transformation matrix and they use the first one as the stitching layer. They preferred this loss signal because, given the assumption, it would be easier to compute and manage compared to the traditional cross entropy loss function over the next token prediction task.
The plasticity with which they are able to "*translate*" the latent space of one model to another, and the limitations highlighted in their work, was the driving force behind this thesis. Particularly, this work focuses on inter-family model stitching, meaning that the task of aligning latent spaces is being done on models trained by different teams, possibly using different training data and with different architectures.

2.2 Activation Steering

Activation steering, while not being directly linked with model stitching, is the second research strand that motivated this work. "*Activation steering*" is an umbrella term that includes all techniques used to modify ("*steer*") the behavior of a large language model directly within the latent space. While the same results achievable with activation steering techniques are probably obtainable by crafting an ad hoc prompt, these techniques are inherently stronger because they operate inside the "*black box*", directly on the latent space. In its simplest form, activation steering is achieved by computing a "*semantic vector*" for a specific sentiment (for example the "*happiness vector*", or the "*anger vector*") and then it is introduced inside the large language model, for

example by adding it to a latent space, to steer the internal reasoning of the model.

Recently, there have been different works on the topic: for example, a framework [1] have been developed to uncensor (i.e bypass their guidelines on sensitive contents) almost any open source model via activation steering. In another blog post on Huggingface [2], activation steering is used to make Llama-3 "*think*" it is the Eiffel Tower. Even if the same results can be achieved by working at a higher level, crafting specific prompts (prompt engineering), these works are inherently more fascinating because they empirically demonstrate that working on the latent space can produce concrete results.

This opens the doors to many different works. The work proposed by Wang et al. [23] demonstrate how injecting the "*truth*" vector inside a model improves its capabilities on benchmarks and reduces hallucinations. Additionally, the work of Thasarathan et al. [20] demonstrates that it is possible to create a Universal Sparse Autoencoder (USAE) that aligns interpretable concepts between different vision models. That is, from one model it is possible to extract one vision concept and feed it to another model under consideration. This work further strengthens the idea that probably an hidden state can be translated from one model to another, keeping coherent generation.

Finally, the work of Bello et al. [2] bridgesactivation steering and model stitching. In their work, they demonstrate that it is possible to "*translate*" a steering vector created for one model to another via a linear transformation. This work in particular was my second source of motivation for my work: what if it were possible to translate a whole hidden state from one model to another?

[1] https://huggingface.co/blog/mlabonne/abliteration

[2] https://huggingface.co/spaces/dlouapre/eiffel-tower-llama

Chapter 3

Background

This chapter establishes the theoretical foundations underpinning this thesis. It provides an overview of the core components of modern Large Language Models (LLMs), ranging from low-level tokenization and embeddings to the high-level Transformer architecture. Furthermore, it introduces a core element for this work, Sparse Autoencoders, that will be central to many experiments involving affine transformations between latent spaces.

3.1 Tokenizers

The tokenizer serves as the interface between raw text and the numerical computation of an LLM. It functions as a deterministic mapping system that converts a string of text into atomic units of text, known as tokens. These tokens are then mapped, via a dictionary or a hash map, to a unique numerical value that can be fed to models, such as large language models. The detokenization works in the opposite direction: a model predicts a numerical indices that are mapped to tokens. Multiple tokens produce a human readable sentence. This tokenization process fundamentally defines the granularity at which a model processes information, as illustrated in Figures 3.1 and 3.2.

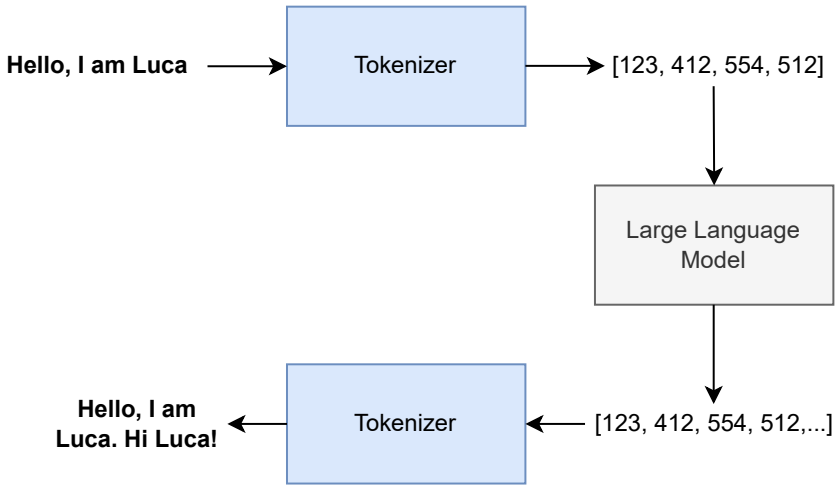

Figure 3.1: The standard pipeline for processing and generating text in an LLM.

Modern tokenizers typically rely on sub-word algorithms, such as Byte-Pair Encoding (BPE) [18], to efficiently handle rare words and morphological variations while maintaining a fixed vocabulary size. However, the specific segmentation rules and vocabulary size vary significantly across model families. For instance, the string '*sunglasses*' might be tokenized as `['sun', 'glasses']` by one model, but as `['sunglasses']` by another. In the context of this work, this possible discrepancy between tokenizers will be problematic. Continuing the previous example, if one model has conceptualized the object as a single token, *'sunglasses'*, but another has internalized the same concepts using two tokens, *'sun'* and *'glasses'*, the alignment gains an additional complexity layer. Instead of being injective as in the related works (in model stitching within the same family the tokenizers of both receiving and giving model are the same, in steering vector translation the steering vector is not related to any token but it is already an abstract object), the mapping function (i.e., the stitching layer) would become surjective, because the latent space of one token maps to possibly more than one different token in the second model latent space. This mismatch necessitates alignment techniques that will be detailed in chapter 4.

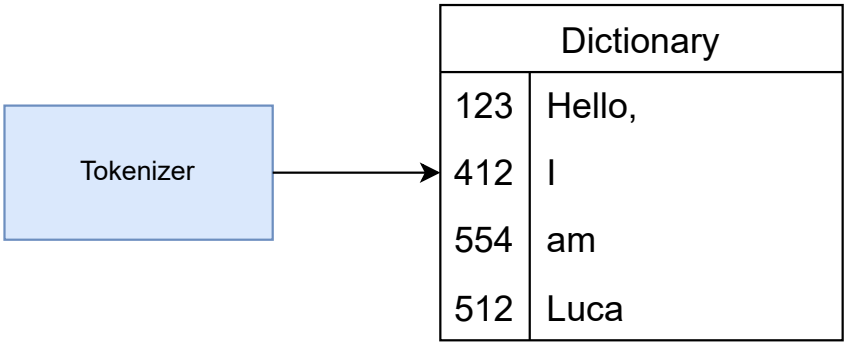

Figure 3.2: The internal structure of a tokenizer represented as a mapping between text units and integer IDs.

3.2 Embeddings

Once tokenized, each discrete token ID is mapped to a continuous, high-dimensional vector known as an *embedding*. These vectors populate the *embedding space*, a geometric space where semantic relationships are encoded as distances and directions.

As illustrated conceptually in Figure 3.3, both position and direction carry meaning. Semantically related terms cluster together, and vector arithmetic can often recover analogical relationships, such as the vector operation $\vec{v}_{\text{king}} - \vec{v}_{\text{man}} + \vec{v}_{\text{woman}} \approx \vec{v}_{\text{queen}}$.

This peculiar property of embedding is called *vector semantics* [3] and it is not unique to large language models. Prior to the advent of transformer-based architectures [22], there were already models capable of creating embeddings starting from tokens or even documents. Outside the mechanistic interpretability field, where these properties are used to study *how* a model process and store informations, these properties are mainly used to do information retrieval tasks.

The concepts behind vector semantics are really important for this work. If these complex relations between vectors could be translated between latent spaces of different models using an affine transformation it would mean that

most probably transformer based models learn an approximation of an universal vector space, and these representations differ only for an affine transformation. This idea of universalization is the same that is behind the semantic hub hypothesis.
In this work the main focus is on trying to achieve a latent space translation between a selection of open weight large language models of different families. The goal is to provide preliminary results towards the aforementioned direction.

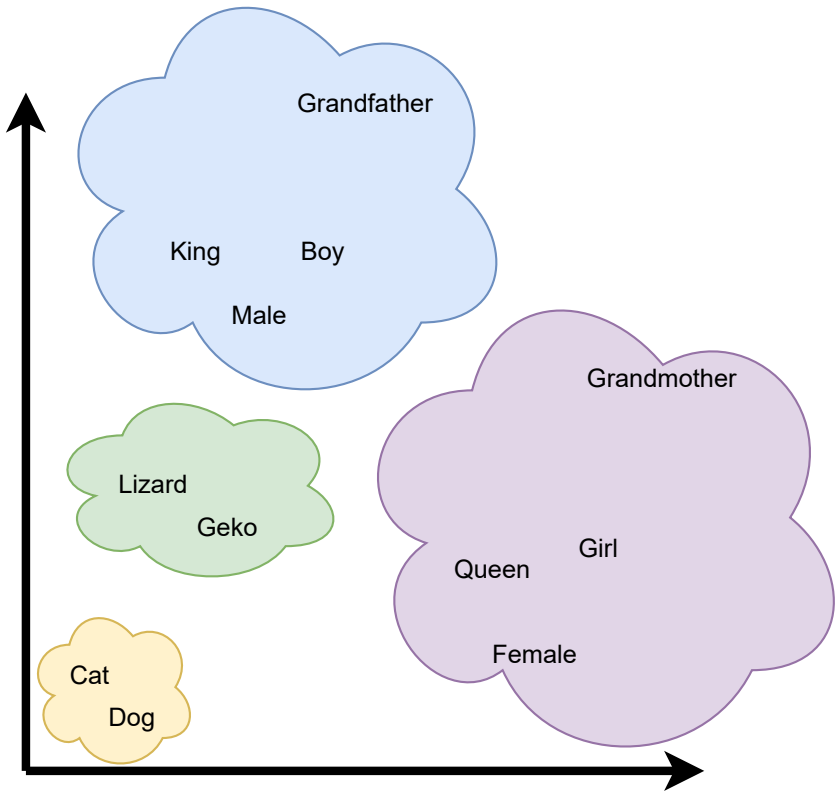

Figure 3.3: A conceptual 2D projection of word embeddings.

3.2.1 The Impact of Training Data

These representations are learned as an emergent property from the training corpus. Consequently, the embedding space mirrors the statistical properties

and biases of the data. When dealing with models within the same family, the training data is most probably the same, because constructing a pretraining dataset for a large language models is very expensive time wise, so the same dataset is used for all models within a generation. This guarantee does not hold when dealing with models from different families, as the training dataset could be different. Unfortunately, this difference is not measurable, because most open weight models are not totally open source, meaning that the source of the data for the training is not publicly available.
This unknown could invalidate the semantic hub hypothesis on which all the subsequently ideas of this thesis are based on: if the sources of the data are *substantially* different (for example, a model trained only on source code and a model trained only on the Italian language), their internal representation will diverge almost certainly. Fortunately, as detailed in depth in Chapter 4, it is safe to assume that large language models trained for general text generation share an overlap in their training data *big enough* to make them converge towards a shared representation.

3.2.2 The Forward Pass and Residual Connections

Understanding the data flow, or *forward pass*, is crucial for comprehending the mechanism of model stitching. The core innovation allowing deep Transformers to train effectively is the **Residual Stream**. Instead of passing the hidden state through a function $x_{l+1} = F(x_l)$, the Transformer employs a residual connection:

$$x_{l+1} = x_l + F(x_l) \tag{3.1}$$

where x_l is the input to layer l, and F represents the processing block (Attention or Feed-Forward Network).
This implies that the hidden state x acts as a shared memory, or ”stream”, that travels through the network, with each layer reading from it and writing a residual update back to it. This additive property suggests that the representations at layer l and layer $l + 1$ share the same vector space and are relatively close to each other. This architectural feature is theoretically what permits identifying a mapping between the residual streams of different models.

3.3 Activation Steering

Activation Steering, or *Activation Engineering* [21], is a technique to control model behavior by intervening in the residual stream. A ”steering vector” is added to the hidden state at a specific layer to bias the generation towards a desired attribute, such as honesty, refusal, or sentiment. Figure 3.4 shows a conceptual representation of how steering vectors are extracted from models.

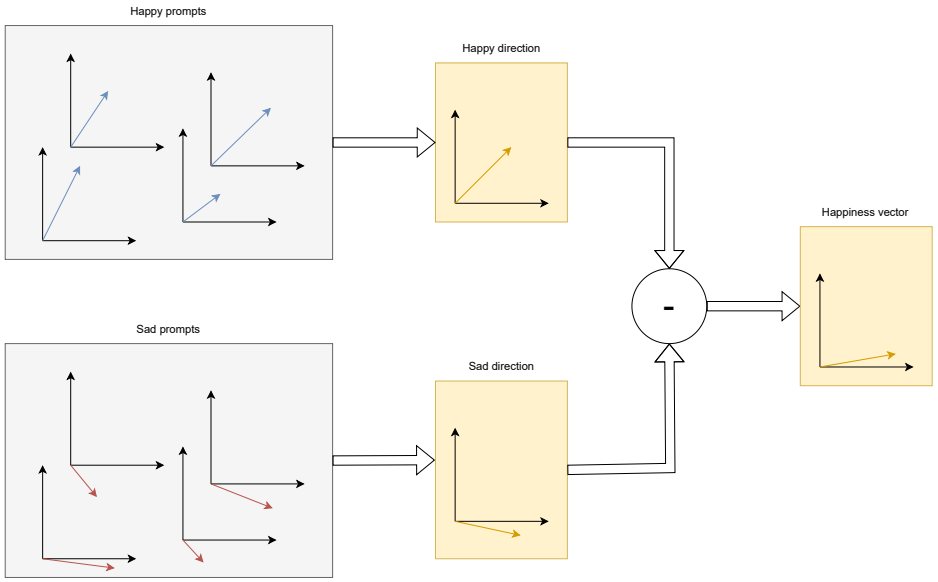

Figure 3.4: Extraction of a steering vector by contrasting activations of opposing concepts.

Recent research [13] has demonstrated that these vectors can be translated between different models, implying that the directions encoding semantic concepts are preserved across architectures. This thesis extends this concept by investigating whether it is possible to translate the entire hidden state rather than a single vector.

3.4 Sparse Autoencoders (SAEs)

Sparse Autoencoders are a primary tool in mechanistic interpretability [19] used to disentangle the *polysemanticity* of neurons. In a standard Transformer, a single neuron may activate for unrelated concepts. This phenomenon, known

as *superposition*, means that a single concept is represented by a distributed linear combination of many neurons.
Conceptually, SAEs map the dense hidden state into a much higher-dimensional, sparse latent space (Figure 3.5). By enforcing a sparsity penalty during training, the SAE forces the model to represent information using a small number of active features.

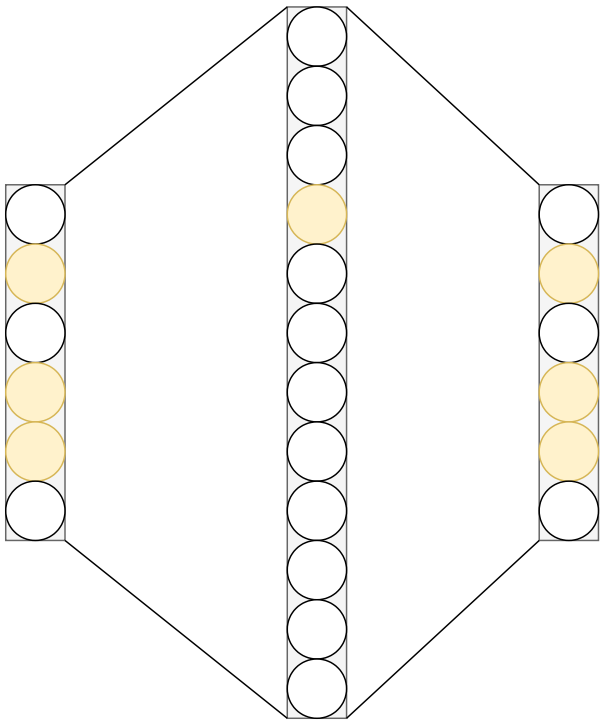

Figure 3.5: Architecture of a Sparse Autoencoder applied to an LLM layer.

These sparse features often correspond to monosemantic concepts, such as a specific feature for ”pointers in C code” or ”references to The Beatles”. Concepts in the latent space of a model are updated layer by layer, with their respective vector undergoing many translations and transformations detailed by the attention and feed forward layer (section 3.2.2). For this reason, the construction of a SAE is local and specific to a single layer.
Training a Sparse Autoencoder is not a simple task, because it must respect two strict criteria: it must be able to reconstruct its input with high fidelity and its internal hidden state must be sparse, in order to be interpretable. To

achieve both task, the training follows two distinct loss functions: the L_1-norm (defined as the sum of the absolute values) of its hidden state ensures sparsity, because solutions that produces distributed high values are penalized. The reconstruction capability is instructed by the L_2-norm of the difference between the input and the reconstructed output. This loss function is called Mean Squared Error (MSE), and it measures how different are two vectors by averaging the sum of the squared difference elements wise.

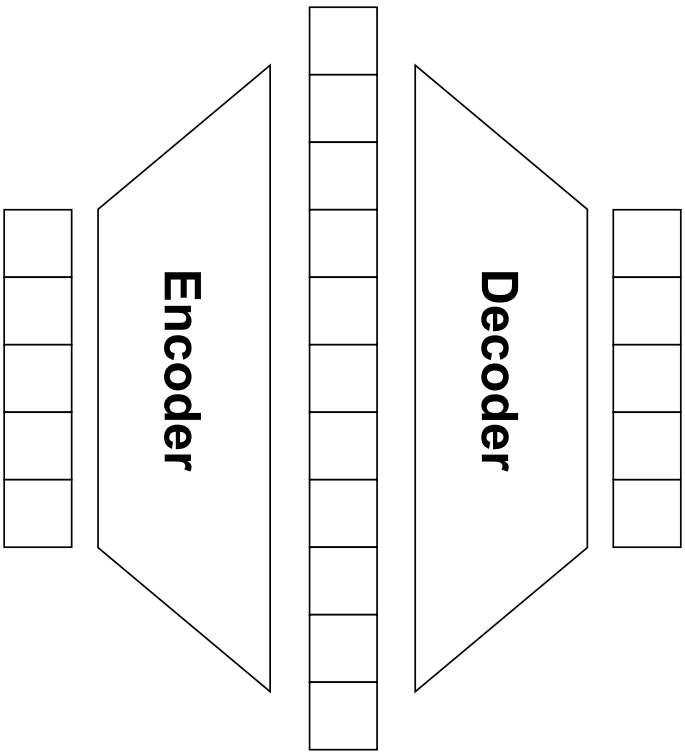

Figure 3.6: Structure of a sparse autoencoder. It is composed by an encoder and a decoder

Figure 3.6 depicts an architectural overview of a sparse autoencoder. It is composed by two parts: an encoder, that project the input into an higher dimensional vector, and a decoder, that downproject the hidden state back to its original dimension. While the decoder is a simple feed forward layer, the encoder is composed by both a feed forward layer and an activation function, specifically the Rectified Linear Unit (ReLU). The ReLU activation function

has a very important role, because it flattens all negative values to zero, while positive values remains untouched. This assures that the sparse autoencoders do not degenerate to an identity function, that would keep a dense nature. With the ReLU and the L_1-norm, negative values do not contribute to its internal representation, and positive values are instructed to be as small as possible. Joined with the L_2-norm signal, that instruct the sparse autoencoder to reconstruct the input, the results is that only few neurons are activated per forward pass.

3.5 The Semantic Hub Hypothesis

The *Semantic Hub Hypothesis* posits that large language models, driven by the structure of natural language and shared training data distributions, converge towards a universal internal representation of concepts.

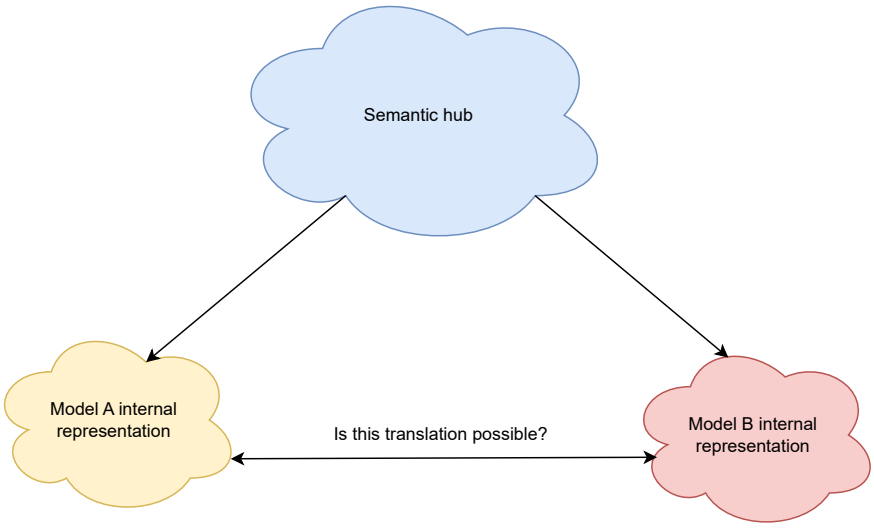

Figure 3.7: A visual representation of how different models might converge toward a similar semantic representation.

While prior work has explored this convergence within model families [4], this thesis investigates whether this universality holds across different architectures, such as between Llama and Gemma. Figure 3.7 schematizes this

idea. Given two LLMs, A and B, if their internal representations can be interpreted as projections of a shared universal semantic space, a natural question arises: is it possible to *translate* the semantic representation of concepts from A into the semantic representation used by B? Moreover, can this translation be achieved through a linear transformation, or at least approximated as closely as possible by one?

The *Stitching Layer* proposed in this work is the mechanism designed to learn such a transformation. Once this mapping is obtained, future work could investigate whether its existence is evidence for the presence of a shared semantic hub, or whether it emerges from other structural properties of the models. Due to time constraints, however, this work focuses solely on empirically demonstrating that such a transformation can be learned, without further investigating the underlying mechanisms that make it possible.

Chapter 4

Inter-family Model Stitching

This chapter introduces the technical details that underlie my work. The goal is to understand whether it is possible to create a stitching layer between models of different families, with different architecture and tokenizer. The intuitions underlying my approach to this task can be summarized as follows, formulated according to the concepts described in Chapter 2 and Chapter 3: *two Large Language Models (LLMs) trained on sufficiently large datasets converge to similar representations* and *the latent space of a model is directly tied to its tokenizer, and different models utilize different tokenizers.* Despite the inability to confirm, for the first intuition, that the training data for some open-weights LLMs are identical, we can assume, particularly for recent models, that the training corpora are at least substantially similar. This is because most open-source and closed-source datasets aggregate data from the same major web sources. Given that high-quality data sources are finite, it is logical to infer that, above a certain scale threshold, there is significant overlap between the training corpora of different models. If this hypothesis holds, the task of translating hidden states between models using a mostly linear transformation becomes feasible, as supported by the work of Chen et al. [4]. The second assumption poses a challenge, because, if true, it would necessary to separate ”raw concepts”, which are hypothesized to be independent of the tokenizer, from their corresponding tokenizer-bound latent space. To achieve this goal, I trained Sparse Autoencoders (SAEs) on specific candidate layers intended for stitching. If the aforementioned hypotheses hold, a simple linear layer should be sufficient to translate the hidden state from one model to

another via this sparse feature space, because the sparse autoencoders is able to extract concepts from their dense representation, and concepts should be alignable.
Unfortunately, as introduced in Chapter 3, SAEs are specific to one layer, and cannot be generalized for all layers of a model. For this reason, given a *starting* model that processes the input and a *ending* that processes the output, the stitching point for all experiments is fixed at a specific depth different from each model. On this stitching point, a SAE is built for both model and a stitching layer is trained between the encoder of the starting model's SAE and the ending model's decoder. Conceptually, this stitching layer will learn how to align concepts extracted from the starting model according to the decoder of the ending model.
The following sections detail the structure of my work, by firstly declaring the hardware constraints I had to follow. Then, Section 4.2 reports all the training procedures I followed to train sparse autoencoders. Section 4.3 introduces the procedure to train a stitching layer between sparse autoencoders of different models. Sections 4.3.1 and 4.3.2 introduce two additional stitching modalities, by changing the the starting assumptions.

4.1 Computational resources

Due to the order of magnitude of LLMs in terms of parameters, and limited hardware resources (restricted to 24 GB of GPU memory), several constraints and premises apply to this work. All training experiments, if not specified, were executed with the following configuration: 2 epochs, a sequence length of 128, 3.6×10^5 training samples, 1.0×10^4 validation samples, a batch size of 4, and a gradient accumulation size of 32, utilizing *bfloat16* precision.
These figures translate to a total of $3.6 \times 10^5 \times 128 = 4.6 \times 10^7$ unique tokens per epoch. Typically, when pre-training LLMs, the order of magnitude for unique tokens is approximately 1.0×10^{12}, which is five orders of magnitude larger than the availability in this setup. Additionally, the training data used for the stitching layer is composed exclusively of English text, whereas the base models are multilingual. Given the training methodology presented later, this setting implies that the stitching layer is restricted to learning only

English token mappings. This constraint actually provides an interesting testing ground for zero-shot cross-lingual capabilities, which will be discussed in a later section. While these factors ensure reproducibility in a domestic hardware environment, they likely pose significant challenges for a comprehensive training cycle and must be taken into consideration when evaluating the results.

4.2 Training Sparse Autoencoders

Despite being increasingly common in mechanistic interpretability, training Sparse Autoencoders (SAEs) presents numerous challenges. There is currently no single standardized architecture or training recipe, and many variants exist. For my experiments, I selected the most standard approach, illustrated in Figure 3.5.
Each SAE is specific to a single layer, constructed with the objective of reconstructing the output of that underlying layer. In addition to the reconstruction loss, calculated as the L2-norm (MSE) between the real and reconstructed output, a sparsity loss (L1-norm) is introduced to force the activation of as few neurons as possible. The complete loss formula is as follows (4.1):

$$\mathcal{L}_{\text{SAE}} = \text{MSE}(y, \hat{y}) + \alpha \|\hat{h}\|_1 \tag{4.1}$$

where y is the output of the underlying LLM layer, $\hat{y}$ is the reconstructed output from the SAE, $\hat{h}$ represents the SAE hidden activations, and α is a coefficient weighting the contribution of the L1 norm.

4.2.1 Training pipeline

Training a sparse autoencoder required a strong multi-step pipeline, due to the task itself and to the hardware requirement. Since a sparse autoencoder is trained to reconstruct an hidden layer of a large language model, two prerequisites are needed: the layer index on which the reconstruction will be done and the specific layer's activation. The layer's index for each model is selected by taking an index at approximately $70\% - 80\%$ of the total depth. For example, if a model has 10 layers, the stitching would be done on the 7-th or 8-th index.

For the activation, a naïve solution would be to load a LLM in memory, compute the forward pass up the layer's index, and then use the layer's output to train the sparse autoencoder. This solution would waste most of the computation in the forward pass of the large language model, and given the hardware constraints, it was infeasible.

Instead, I precomputed and stored the layer's hidden state prior to the SAE training. In particular, I stored $30'000$ activations with a sequence length of 128 obtained from the *OpenWebText* [6] dataset. The stored activations correspond to $30'000 \times 128 = 3'840'000$ tokens, but since they are latent spaces and not activations, assuming for sake of simplicity an hidden size of 2048 per models and `fp16` precision, to store all the activations $3'840'000 \times 2048 = 7'864'320'000$ numbers $= 15'728'640'000$ bytes ≈ 15 GB are needed. For an hybrid models, two SAEs are required (one for the model in the first section and one for the model in the second section), increasing the disk usage to 30 GB. Assuming two models with 20 layers each, 40 SAEs would be required to test all possible stitching combinations. This would require $40 \times 20 = 800$ GB only to store the activations needed to train each SAE. This was unfeasible, both time wise and storage wise. For this reason, I decided to test my experiments only on one layer per model, meaning that the stitching point is fixed for every experiments. Table 5.1 details the stitching layer for every layer considered in my experiments.

To ensure that a sparse autoencoder is trained correctly, I created a validation split containing the 25% of the training data. This validation set was used to compute two metrics: *average L_0-norm* and *average explained variance*. The total explained variance is computed using the equation 4.2 and it is averaged across the number of batches

$$\text{explained variance} = 1 - \frac{MSE}{Var(y)} \tag{4.2}$$

A sparse autoencoder is considered valid only if it has an average L_0-norm below 0.20 and an average explained variance above 0.90. This design choice was employed because a good sparse autoencoder should be sparse (low L_0-norm), and most importantly must be able to reconstruct its input (high explained variance). The threshold values have been found empirically, because there exists a tradeoff between the two dimensions: a sparser model would

have a lower explained variance, and vice versa. These threshold ensured that the sparse autoencoders were good enough for the task.
During my experiments, I observed high sensitivity to training hypeparameters: SAE expansion factor, batch size, warmup steps, learning rate and L_1 coefficient. Initially, to ensure the best combination as possible, I ran a grid search over different combinations of there hyperparameters, but the resulting search space turned out to be too big to be explored in usable time. For this reason, I decided to fix some hyperparameters, while others would be ran over a grid search: the SAE expansion factor was fixed to 8, this means that the hidden size of a sparse autoencoder is $8\times$ the hidden size of the large language model. The batch size was set to 128 and the warmup was set to 5% of the total training steps. The learning rate was selected between $5 \times 10^{-5}, 5 \times 10^{-4}$ and 1×10^{-4} while the L_1 coefficient varied between $1 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-5}, 5 \times 10^{-6}, 1 \times 10^{-6}$ and 5×10^{-7}.
This grid search yields different SAE that are within the imposed thresholds. These potential SAE are then ordered by a score function defined in equation 4.3

$$\text{score} = \alpha L_0 - \beta Var \tag{4.3}$$

where α weights the L_0 norm and β weights the explained variance. Varying α and β modify how important is one loss with respect to the other. All SAEs are then sorted in ascending order, and the first one is selected. Since for the goal of the task, having an higher explained variance is more important than to have a sparser SAE, I set $\alpha = 1.0$ and $\beta = 2.5$. These values were chosen empirically, and they align really well with what I would have chosen manually.

4.2.2 Additional finetuning

For the sake of training a sparse autoencoder, the previous pipeline covers all the requisite and should produce an efficient SAE. Despite this, for this work, the sparse autoencoder is employed as a substitute for its underlying layer, because the stitching will be made between two sparse autoencoders, and each sparse autoencoder must be able to ensure that its output is good

enough to replace the output of the underlying layer. The explained variance is not enough, because mathematical similarity in this context does not necessarily means functional similarity. For these reason, to finalize the training, a second stage is performed in which the SAE layer fully replaces the underlying layer within the model. The loss signal is derived from the LLM's reconstruction error (Cross-Entropy Loss) on the next-token prediction task. This stage is conducted with a fixed learning rate of 5×10^{-5}. While it is highly beneficial for decreasing the model's perplexity, it often slightly degrades sparsity and reconstruction fidelity, though they remain above the established thresholds.

4.3 Stitching in the SAE Space

Once that SAEs are trained, I concentrated my work in making a functional hybrid model. This section details how a stitching layer can be trained between sparse autoencoders of differend models, in order to obtain an hybrid model. In this architecture, the input sequence is processed by the first model (Model A) up to the target layer. It is then up-projected by SAE A's encoder. The affine stitching layer processes this sparse representation, which is subsequently down-projected by SAE B's decoder before processing continues in the second model (Model B). The hypothesis is that if the high-level concepts of two LLMs differ only by an affine transformation, joining two SAE layers via a linear map should be feasible, as the stitching layer aligns the high-level semantic features. Figure 4.1 illustrates the architecture and the stitching process.

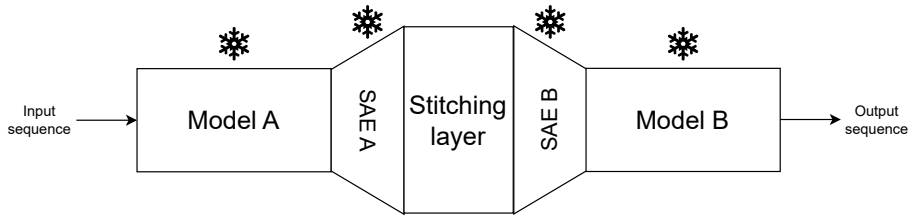

Figure 4.1: Architectural overview of the stitching process over SAE layers.

To train an hybrid model, I employed the standard next-token prediction task used in traditional LLM pre-training. The error signal generates at the second model's head and backpropagates directly to the stitching layer, as all other

weights are frozen. This training objective differs from that employed by Chen et al. [4], who used a reconstruction-specific loss under the assumption that the translation between hidden states is reversible. Given that reversibility may not hold when employing SAE layers, due to the non-linearities and information filtering inherent in sparse encoding, I opted to train directly on next-token prediction.

Aligning Datasets

A critical aspect of inter-family model stitching is the necessity of aligning datasets. Empirically, I found that training on a general English dataset resulted in extremely poor performance for the stitched model (in terms of Cross-Entropy Loss and completion quality), probably due to the concepts explained in section 3.1. I resolved this by filtering the dataset to retain only sequences that are tokenized into the same set of tokens by both tokenizers and, crucially, yield the same text when de-tokenized.
For example, the compound word ”milestone” might be segmented differently by two tokenizers. The first check ensures that the sequence length is identical for both; for instance, a length of 2 vs. 2 passes, but different lengths fail. If this assertion holds, the next step is to check whether the text is tokenized into the same tokens. For example, if both tokenizers yield the segments [*”mile”*, *”stone”*], the check passes. However, if one yields [*”milest”*, *”one”*] and the other [*”mile”*, *”stone”*], the check fails, because the word is tokenized differently by the two tokenizers.
This imposes strong limitations on the dataset:

- **Tokenizer Assumption:** There is no guarantee that two different tokenizers share identical tokens, even at different indices. Fortunately, for the models selected, sufficient overlap exists to create an aligned dataset, probably because tokenizers are trained on similar corpora of data.

- **Sequence Length constraint:** The longer the sequence, the rarer this alignment becomes. I found that a sequence length of 32 yielded a

good tradeoff between context and the percentage of usable aligned sequences. Reducing the length from 128 to 32 further complicates training by limiting context. Despite this additional limitation, the results remain promising.

I term this constraint *temporal alignment*. In practice, I am not modifying the tokenizers to force alignment but rather selecting a subset of data that naturally satisfies the requirement of being "consumed" at the same rate by both models. We consider the time dimension here as the reading direction and the implicit ordering of words. If two tokenizers feed the network at the same pace, the n-th latent vector of Model A corresponds temporally to the n-th latent vector of Model B. Without this, the stitching layer faces a dual burden: translating the semantic concepts *and* managing a temporal shift (e.g., aligning token 5 of Model A with token 7 of Model B). By providing an aligned dataset, the temporal mismatch is removed, allowing the stitching layer to focus solely on semantic translation.

Given the aforementioned considerations, for each pair of tokenizers, corresponding to a pair of models used to assemble an hybrid model, an aligned dataset is built considering $360'000$ record from OpenWebText [6] with a sequence lenght of 32. The filtering process resulted in a $30\% - 40\%$ data loss, meaning that only $70\% - 80\%$ of the records were alignable. Unfortunately, increasing the sequence length drastically increased the data loss during the filtering process, and 32 was found as a tradeoff between sequence length and data retainment.

While filtering data to avoid the root cause of misalignment is indeed a limitation, it allows to determine if the fundamental task of cross-family stitching is possible under ideal conditions. Generalizing this to unaligned sequences is a prominent direction for future work.

4.3.1 One-block Direct Stitching

The previous section detailed the procedure for training a stitching layer between two frozen, pre-trained SAE layers. While methodologically sound and strict with respect to the "semantic hub" hypothesis, we can relax some assumptions to simplify the process.

If we treat the encoder of the first model's SAE, the stitching layer, and the decoder of the second model's SAE as a single composite block, the optimization problem changes. In this "One-block" approach, the stitching mechanism gains a source of non-linearity, as the encoding part of the SAE retains its ReLU activation (see Figure 4.2).

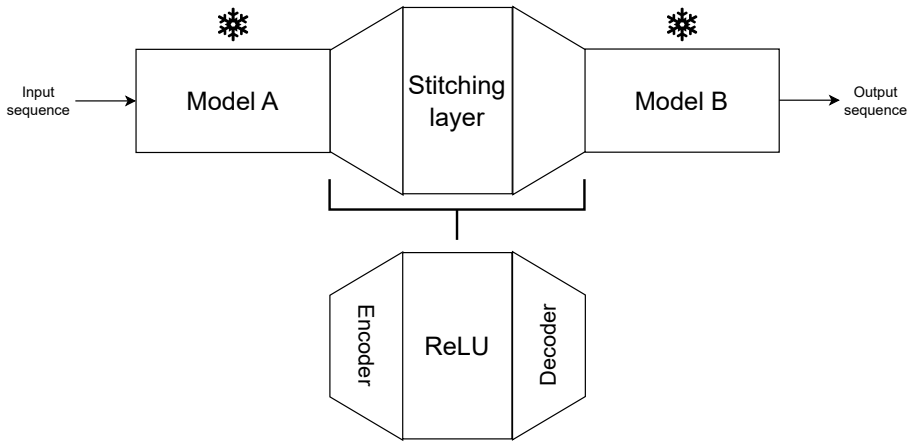

Figure 4.2: Representation of how one-block stitching works.

While introducing non-linearity makes the resulting transformation less interpretable (as it is no longer strictly affine), this training procedure is significantly easier to manage, as it removes the need to train SAE layers independently. This opens the door to testing multiple stitching positions efficiently. However, to maintain comparability with the independent SAE approach, I did not investigate multiple positions further in this thesis.
Including this modality, there are four comparisons: with and without aligned datasets, and with independent SAE layers versus the direct one-block approach. Additionally, I investigated a fifth modality: utilizing the same tokenizer for both the input and output models, despite their architectural differences.

4.3.2 Single Tokenizer Stitching

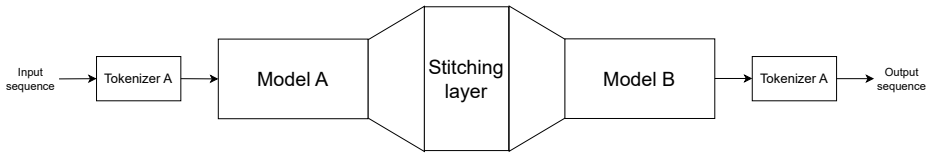

Figure 4.3: Single tokenizer stitching scheme. The input tokenizer (Model A) is used to de-tokenize the output logits generated by Model B.

This setup, discovered by chance via a configuration error, involves de-tokenizing the output sentence using the *input* model's tokenizer, even though the *output* model belongs to a different family with a distinct vocabulary size and dictionary.
For this setup to function without errors, a strict mathematical requirement must be met: the vocabulary size of the output model must be greater than or equal to the vocabulary size of the input model. This is necessary because, when computing the Cross-Entropy Loss during training, the target labels (derived from the input model's tokenizer) must be valid indices within the output model's logit vector. If the output vocabulary is larger, this requirement is always satisfied.
This works due to the properties of the indexing operation in the loss calculation. Consider the Cross-Entropy formula:

$$\text{CE} = -\sum_{i} \hat{P}(i) \log P(i) \tag{4.4}$$

Since the target distribution $\hat{P}$ is one-hot encoded, this simplifies to:

$$\text{CE} = -\log(P(\text{target\_index})) \tag{4.5}$$

Let P be the logits of the output model (size 6) and $\hat{P}$ be the labels from the input model (size 3).

$$P = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.3 \\ 0.1 \\ 0.1 \end{bmatrix}, \quad \hat{P}_{\text{input}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \implies \text{target\_index} = 0 \tag{4.6}$$

Mathematically, the vectors have different dimensions. However, the implementation calculates the loss by looking up the logit at the target index provided by the label. Since the index (0) exists in the larger vector P (indices 0 to 5), the operation is valid:

$$\text{Loss} = -\log(P[0]) = -\log(0.1) = 2.30 \tag{4.7}$$

Conversely, if the input vocabulary were larger than the output vocabulary, the target index could exceed the bounds of P, causing an index error.

While this explains *why* the computation is possible, conceptually it is difficult to grasp what the stitching layer is learning. The stitching layer must map the internal representation of Model A to a representation for Model B such that Model B produces a logit distribution where the maximal value corresponds to an index valid in Model A's vocabulary. This implies that the stitching layer is forcing Model B to act as if it was built to work for Model A's tokenizer.

In this context, the Cross-Entropy Loss value loses its standard interpretation. The output logits are distributed over a larger space (V_{out}) than the relevant targets (V_{in}). Since $V_{out} > V_{in}$, a significant portion of the probability mass may be assigned to indices that represent "impossible" tokens for the input tokenizer, rendering them mathematically irrelevant to the specific target but creating noise in the distribution.

During inference, this setup does not have a hard constraint preventing the output model from generating an index $i > |V_{in}|$. If this occurs, decoding with the input tokenizer fails. However, surprisingly, for in-distribution generation (e.g., common English sentences), the hybrid model is able to produce tokens within the valid range of the input tokenizer and maintains context.

This setup, while theoretically unorthodox, suggests that the an alignment between latent spaces may allow for forcing a model to operate within an alien

token space purely via activation steering. A more in-depth discussion follows in later chapters, after presenting benchmarks and ablation studies for all setups.

Chapter 5

Results and benchmarks

In Chapter 4, I introduced six distinct stitching strategies. These can be categorized into stitching within the SAE dimension and One-block stitching, with a further variant involving the use of a unified tokenizer for both models. This chapter presents the quantitative results achieved, analyzed in terms of Cross-Entropy Loss, where applicable, and standard benchmarks such as HellaSwag [25] and Massive Multitask Language Understanding (MMLU) [8]. For the experiments involving open-weights Large Language Models (LLMs), I selected Gemma-2-2B by Google, Llama-3-8B by Meta, and Qwen2.5-7B / Qwen2.5-3B by Qwen. The stitching layer's for each model, and the correspective vocabulary size, is reported in table 5.1:

Model	Stitching Layer	Vocabulary Size
Qwen/Qwen2.5-7B	20	151'665
google/gemma-2-2b	20	256'128
meta-llama/Meta-Llama-3-8B	25	128'256
Qwen/Qwen2.5-3B	27	151'665

Table 5.1: Model names and corresponding stitching points and vocabulary size.

The models were paired following the constraints outlined in previous sections: the vocabulary size of the second model (the receiver) must be larger than that of the first, and the parameter count of the first model (the sender) should ideally be higher to ensure sufficient representational density. Table 5.2 details the pairs that compose the three hybrid models I created. Each hybrid model is tested with all six stitching modalities.

First Model	Second Model
Qwen/Qwen2.5-7B	google/gemma-2-2b
meta-llama/Meta-Llama-3-8B	google/gemma-2-2b
meta-llama/Meta-Llama-3-8B	Qwen/Qwen2.5-3B

Table 5.2: Model's pair composing the hybrid model under examination.

Model	**Avg CE Loss**
Llama-3-8B (Meta)	2.1853
Gemma-2-2B (Google)	2.4173
Qwen2.5-3B (Qwen)	2.2308
Qwen2.5-7B (Qwen)	2.1987

Table 5.3: Baseline performance of Vanilla models on the test set.

5.1 Cross-Entropy Loss

Cross-Entropy (CE) loss serves as the standard training objective for Large Language Models. It indirectly measures the model's *surprise* when encountering the ground truth label given its predicted distribution. This metric is intrinsically linked to *perplexity* (PP), defined as:

$$PP = e^{H}$$

where H represents the Cross-Entropy loss. Due to this exponential relationship, even a marginal increase in Cross-Entropy results in a significant rise in perplexity.

While CE loss is typically a robust metric for evaluating model performance, it is not universally applicable in this context. Specifically, for the stitching methodology utilizing a unified tokenizer, the metric loses its standard mathematical interpretation, as discussed in Chapter 4. Nevertheless, for compatible setups, CE loss allows for a direct comparison with the findings of Chen et al. [4].

To establish a baseline for comparison, I computed the CE loss of the original, unmodified models (Table 5.3) on the same test set, which consists of a subset of *The Pile* [5]. These baseline values are essential for quantifying the performance degradation in the stitched models and benchmarking the results against reference literature.

5.1.1 SAE Space Stitched Models

Table 5.4 presents the performance of hybrid models where the stitching layer is trained as an affine transformation between the encoder's SAE of the first model and the decoder's SAE of the second model. In this setup, all weights are frozen except for the stitching layer.
The results distinguish between models trained on a standard dataset versus those trained on an aligned dataset.

Model Combination	Aligned Dataset	Avg CE Loss
Non-aligned models		
Llama-3-8B – Qwen2.5-3B	False	6.7188
Llama-3-8B – Gemma-2-2B	False	7.1140
Qwen2.5-7B – Gemma-2-2B	False	6.3263
Aligned models		
Llama-3-8B – Qwen2.5-3B	True	3.2633
Llama-3-8B – Qwen2.5-3B	False	10.4389
Llama-3-8B – Gemma-2-2B	True	3.2716
Llama-3-8B – Gemma-2-2B	False	10.7030
Qwen2.5-7B – Gemma-2-2B	True	3.2597
Qwen2.5-7B – Gemma-2-2B	False	11.3733

Table 5.4: Performance comparison of Hybrid SAE models with and without dataset alignment.

The data demonstrates that dataset alignment significantly boosts performance. For models trained on a standard (non-aligned) dataset, the loss remains above 6.32 across all combinations, corresponding to a perplexity of approximately 545. This implies that, for any given prediction, the model is uncertain among 545 different tokens. In contrast, aligned models achieve a Cross-Entropy loss around 3.26, corresponding to a perplexity of 26, a drastic improvement in predictive certainty.

Model Combination	Reference Vanilla	Avg CE Loss	Degradation (%)
Llama-3-8B – Qwen2.5-3B	Qwen2.5-3B	3.2633	+46.3%
Llama-3-8B – Gemma-2-2B	Gemma-2-2B	3.2716	+35.4%
Qwen2.5-7B – Gemma-2-2B	Gemma-2-2B	3.2597	+34.8%

Table 5.5: Performance degradation of aligned hybrid models relative to vanilla backbones.

Table 5.5 quantifies the performance degradation relative to the vanilla reference model (which corresponds to the second model in the hybrid pair). The observed drop in performance ranges between 34.5% and 46.3%. These figures are comparable to the results reported by Chen et al. [4] when stitching a smaller model into a larger one. Considering the challenges inherent to this work, namely the use of different model families, limited computational and data resources, and the absence of fine, tuning for stitching layer placement—these results are highly promising. Further implications of these findings are discussed in Chapters 6 and 6.3.

5.1.2 One-block Stitched Models

This section details the performance of models stitched using the *One-block* strategy, where the stitching assembly comprises the encoder, the linear stitching layer, and the decoder as a single composite unit. The results are summarized in Tables 5.6 and 5.7.

Model Combination	Aligned Dataset	Avg CE Loss
Non-aligned models		
Llama-3-8B – Qwen2.5-3B	False	4.4885
Llama-3-8B – Gemma-2-2B	False	7.2558
Qwen2.5-7B – Gemma-2-2B	False	5.3960
Aligned models		
Llama-3-8B – Qwen2.5-3B	True	3.2338
Llama-3-8B – Qwen2.5-3B	False	10.3898
Llama-3-8B – Gemma-2-2B	True	3.1400
Llama-3-8B – Gemma-2-2B	False	10.7703
Qwen2.5-7B – Gemma-2-2B	True	3.2270
Qwen2.5-7B – Gemma-2-2B	False	10.4007

Table 5.6: Cross-Entropy Loss results for One-block stitched models across aligned and non-aligned datasets.

The One-block methodology yields marginally superior results compared to the SAE space stitching approach. This improvement is expected, as this architecture grants the stitching layer greater degrees of freedom to learn the required mapping. However, viewed from a different perspective, the fact

Model Combination	Reference Vanilla	Avg CE Loss	Degradation (%)
Llama-3-8B – Qwen2.5-3B	Qwen2.5-3B	3.2338	+45.0%
Llama-3-8B – Gemma-2-2B	Gemma-2-2B	3.1400	+29.9%
Qwen2.5-7B – Gemma-2-2B	Gemma-2-2B	3.2260	+33.5%

Table 5.7: Performance degradation of One-block hybrid models relative to the vanilla backbone baselines.

that the gain is minimal suggests that the added complexity is not strictly necessary; the task can be effectively addressed with a simple affine layer, thus reinforcing the core research hypothesis regarding the linearity of the feature space.

5.1.3 Same Tokenizer Stitched Models

Finally, this section reports the results for hybrids trained using a unified tokenizer for both the input and output models. As previously noted, Cross-Entropy loss loses its strict mathematical interpretation in this context; however, it remains useful for relative comparisons. The results are presented in Table 5.8.

Model Combination	Avg CE Loss
One-block stitching	
Llama-3-8B – Qwen2.5-3B	6.2200
Llama-3-8B – Gemma-2-2B	7.3191
Qwen2.5-7B – Gemma-2-2B	7.1525
Stitching in the SAE space	
Llama-3-8B – Qwen2.5-3B	6.5878
Llama-3-8B – Gemma-2-2B	7.6225
Qwen2.5-7B – Gemma-2-2B	7.5083

Table 5.8: Cross-Entropy Loss comparison for Single Tokenizer stitching configurations.

Consistent with previous findings, models trained with One-block stitching perform slightly better than those stitched solely via an affine layer. Even if the metric itself is not fully interpretable in absolute terms, it allows for a *relative* comparison of the same hybrid architecture, as the principle that ”lower cross entropy implies a better model” holds valid.

The following section evaluates these models on standardized benchmarks, enabling a comprehensive comparison of this family of stitched models against the others.

5.2 HellaSwag

HellaSwag is a challenging benchmark designed to evaluate commonsense natural language inference. It is formulated as a multiple-choice task where the model must complete a given context by selecting the most plausible continuation among four options.
To evaluate the models, the standard likelihood-based approach was employed. For each question, the context is concatenated with each of the four possible endings. The model computes the Cross-Entropy Loss over the tokens corresponding to the ending; the option yielding the lowest loss is selected as the predicted answer. The primary metric is accuracy, defined as the ratio of correctly predicted completions to the total number of samples.

5.2.1 Evaluation Methodology for Hybrid Models

For configurations utilizing a single tokenizer (Vanilla models and ”Same Tokenizer” hybrids), the evaluation follows the standard procedure described above. However, hybrid models involving distinct tokenizers for the input (Model A) and output (Model B) introduce a significant challenge: *token misalignment*.
Since the two tokenizers possess different vocabularies and segmentation rules, the sequence length of the tokens generated by Tokenizer A (used to produce input logits) often differs from the sequence length of the target labels generated by Tokenizer B. To address this, the evaluation pipeline was adapted as follows:

1. The full sequence (Context + Ending) is tokenized by Tokenizer A and fed into the hybrid model to generate logits.

2. The logits corresponding specifically to the ”Ending” are extracted based on the length of the context in Tokenizer A’s space.

3. The ground truth ”Ending” text is separately tokenized by Tokenizer B to generate the target labels.

4. Due to the potential length mismatch between the predicted logits (Model A) and target labels (Model B), both sequences are truncated to the minimum common length: $L = \min(\text{len}_{\text{logits}}, \text{len}_{\text{labels}})$.

5. The Cross-Entropy Loss is computed on these aligned subsequences.

This methodology ensures that a comparable loss metric can be derived even when the semantic content is distributed across a different number of tokens.

5.2.2 Results

Table 5.9 establishes the baseline performance of the unmodified (vanilla) models. Since the task involves four choices, a random guess yields an accuracy of 25%.

Model	Accuracy (%)
Meta-Llama-3-8B	78.28
Gemma-2-2B	54.11
Qwen2.5-7B	76.75
Qwen2.5-3B	70.79

Table 5.9: Baseline accuracy of Vanilla models on the HellaSwag validation set.

The results for the various stitching configurations are presented in Table 5.10. The experiments compare three main approaches: One-block stitching, Stitching in the SAE space, and the Single Tokenizer variant, both with and without dataset alignment.

5.2.3 Analysis

The results corroborate the trends observed in the Cross-Entropy analysis. Models trained on aligned datasets consistently outperform their non-aligned counterparts. Similarly, the One-block stitching strategy yields slightly superior results compared to stitching strictly within the SAE space, likely due to the additional flexibility provided by the encoder’s non-linearities.

A notable highlight is the performance of the *Llama-3-8B* $\rightarrow$ *Qwen2.5-3B* hybrid model using One-block stitching on an aligned dataset. It achieves an accuracy of 65.99%, which approaches the vanilla Qwen2.5-3B performance of 70.79%. This combination generally outperforms the others, which can be attributed to the stronger baseline capabilities of the Qwen2.5-3B receiver compared to the Gemma-2-2B receiver.
Interestingly, the configurations utilizing the "Same Tokenizer" strategy achieve accuracies significantly above chance (ranging from 35% to 48%), despite the theoretical mismatch described in previous chapters.
It is worth noting a seeming contradiction: while the absolute Cross-Entropy loss for these hybrid models is often high (as seen in Chapter 5), their accuracy on HellaSwag remains relatively robust. This discrepancy arises because HellaSwag is a *discriminative* task based on relative ranking. Even if the model's output distribution is noisy or has high entropy (leading to high absolute loss), as long as the model assigns a *relatively* lower loss to the correct completion compared to the distractors, it will answer correctly. This suggests that while fine-grained probability calibration is degraded by stitching, the preservation of relative semantic plausibility is maintained.
This reasoning extends to the MMLU benchmark, discussed in the following section.

5.3 Massive Multitask Language Understanding

While the previous benchmark empirically measured the models' ability to perform commonsense reasoning and maintain syntactic coherence, the Massive Multitask Language Understanding (MMLU) benchmark evaluates the depth of knowledge and problem-solving capabilities. MMLU consists of multiple-choice questions covering 57 distinct tasks, ranging from elementary mathematics and US history to computer science, law, and medicine.
Unlike generative tasks, MMLU is evaluated by providing the model with a prompt containing the question and four possible answers (labeled A, B, C, D). The model is required to predict the next token, and the answer is selected by comparing the probabilities assigned to the tokens corresponding to the four labels. The option with the highest likelihood is chosen as the prediction.

Additionally, the model is provided with five examples picked randomly from the categories to help the model understand the nature of the task (5-shot testing). The questions and answers provided are not considered for the accuracy calculation. Since there are four options, a random baseline yields an accuracy of 25%.

This benchmark is particularly well-suited for cross-family model stitching evaluation because the target tokens (*"A"*, *"B"*, *"C"*, *"D"*) are represented by single tokens across all tokenizers. This remove the token misalignment issues observed in generative tasks, allowing for a cleaner assessment of semantic transfer.

The results are reported as the overall accuracy averaged across all categories. Additionally, Figure 5.1 provides a heatmap illustrating performance aggregated by subject area (*STEM*, *Humanities*, *Social Sciences*, and *Other*).

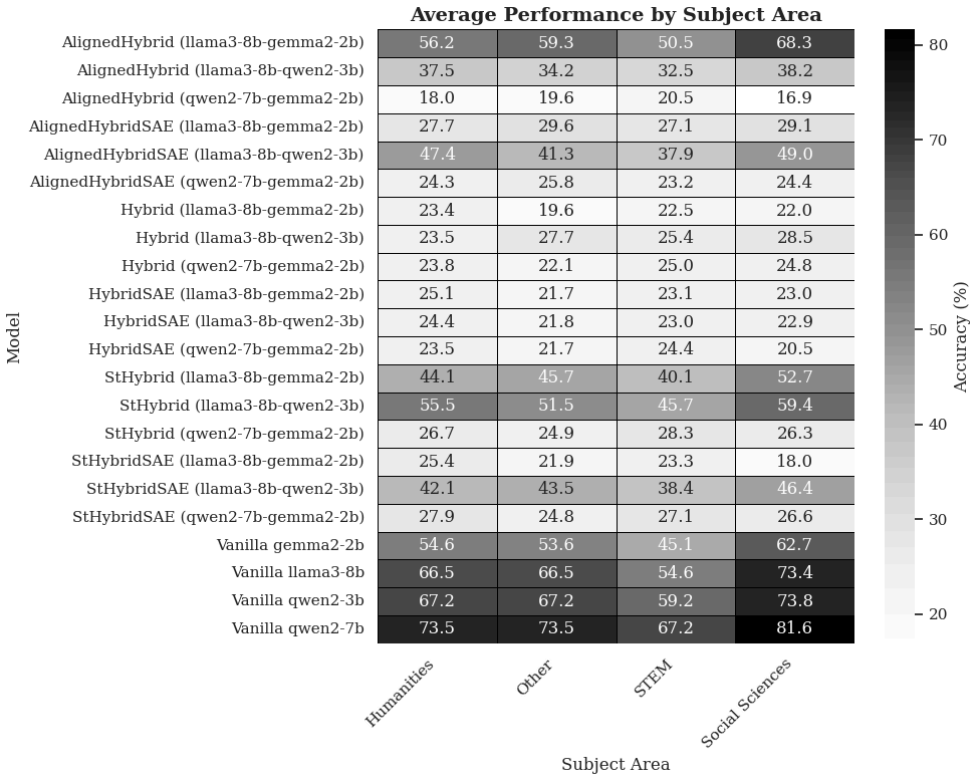

Figure 5.1: MMLU benchmark results, divided by subject area. The prefix *St* indicates models using the Same Tokenizer stitching strategy, while *Aligned* indicates models trained on the temporally aligned dataset.

5.3.1 Results

Table 5.11 establishes the baseline performance of the vanilla models.
Table 5.12 summarizes the performance of the various stitching configurations. The results highlight a distinct separation between models trained on aligned datasets versus standard datasets, as well as the efficacy of the One-block approach over the SAE-space approach for this specific task.

5.3.2 Analysis

The MMLU results offer a more nuanced view of the models' capabilities compared to HellaSwag. While HellaSwag performance was relatively robust even with noisy models, MMLU demands precise knowledge retrieval. Consequently, non-aligned models (both One-block and SAE-space) perform at or below random chance (25%). This confirms that without dataset alignment or a forced tokenizer constraint, the semantic mapping is insufficient for complex question answering.
Notably, only combinations utilizing `Llama-3-8B` as the source model achieved performance significantly above random chance: specifically, `Llama-3` $\rightarrow$ `Gemma-2` and `Llama-3` $\rightarrow$ `Qwen2.5`. In stark contrast, the combination `Qwen2.5` $\rightarrow$ `Gemma-2` consistently failed to outperform random baselines across all stitching modalities.
The underlying cause of this discrepancy remains unclear. It suggests a potential structural or representational incompatibility between the latent spaces of Qwen-2.5 and Gemma-2, which does not manifest when Llama-3 acts as the source. This implies that Llama-3 might possess a more "universal" or accessible representation space, acting as a more effective donor. A comprehensive ablation study across all permutations and modalities would be required to isolate the root cause, representing a time-consuming but essential direction for future research.

5.3.3 Qualitative Analysis: Model Inference

To qualitatively assess the generative capabilities of the stitched models, I collected text completions produced by the `Llama-3-8B` $\rightarrow$ `Qwen2.5-3B` hybrid model. This specific pair was selected for its superior performance on

benchmarks. The generation was performed using the ”Single Tokenizer” setting (using the source tokenizer for output decoding), as it simplifies the inference pipeline by avoiding the need for intermediate detokenization and re-tokenization.
It is important to note that this configuration lacks a hard constraint forcing the target model (Qwen) to predict valid logits for the source tokenizer (Llama). Consequently, the model occasionally predicts out-of-vocabulary tokens, leading to inference crashes. However, successful generations demonstrate both factual recall and syntactic coherence. Table 5.13 reports four examples of different prompts and their completition.
While the model exhibits instability (as seen in the final example with broken HTML/garbage text) and cannot be considered production-ready, these examples provide empirical evidence that the stitching layer effectively translates the semantic intent and factual knowledge from one model's latent space to another's.

Zero-Shot Cross-Lingual Transfer

This subsection examines the `Qwen2.5-7B` $\rightarrow$ `Gemma-2-2B` configuration, specifically using aligned datasets and stitching strictly within the Sparse Autoencoder (SAE) space. This modality was chosen as it aligns most rigorously with the initial theoretical premises of the Semantic Hub hypothesis.
A particularly intriguing property observed is the zero-shot cross-lingual capability. The prompts were constructed in English, but with a pivotal keyword substituted by its Chinese character equivalent (provided below with translations for clarity). The goal was to test if the ”concept” encoded by Qwen (a model with strong Chinese capabilities) could be correctly translated by the stitching layer into a representation that Gemma (an English-centric model) decodes as the correct English continuation. Table 5.14 contains the results.
Although the model does not capture the context in every instance, the successful cases are remarkable. They suggest that the hidden state translation occurs at a conceptual level, abstracting away from the specific language of the input tokens. This supports the theory that LLMs converge on a language-independent ”mentalese” or shared semantic space. The fact that the source model (Qwen) is specialized in Chinese likely facilitates the encoding of these

characters, but the correct decoding by Gemma confirms successful cross-model semantic transfer.

Model Combination	Accuracy (%)
One-block stitching	
Llama-3-8B – Gemma-2-2B	29.90
Qwen2.5-7B – Gemma-2-2B	29.60
Llama-3-8B – Qwen2.5-3B	49.41
Aligned one-block stitching	
Llama-3-8B – Gemma-2-2B	38.78
Qwen2.5-7B – Gemma-2-2B	35.21
Llama-3-8B – Qwen2.5-3B	65.99
Same tokenizer stitching	
Llama-3-8B – Gemma-2-2B	44.07
Qwen2.5-7B – Gemma-2-2B	35.44
Llama-3-8B – Qwen2.5-3B	48.53
Stitching in the SAE space	
Llama-3-8B – Gemma-2-2B	29.27
Qwen2.5-7B – Gemma-2-2B	28.87
Llama-3-8B – Qwen2.5-3B	45.97
Aligned stitching in the SAE space	
Llama-3-8B – Gemma-2-2B	37.54
Qwen2.5-7B – Gemma-2-2B	33.72
Llama-3-8B – Qwen2.5-3B	65.25
Same tokenizer stitching in the SAE space	
Llama-3-8B – Gemma-2-2B	42.41
Qwen2.5-7B – Gemma-2-2B	33.81
Llama-3-8B – Qwen2.5-3B	44.96

Table 5.10: Accuracy comparison across different stitching configurations.

Model	Accuracy (%)
Meta-Llama-3-8B	62.50
Gemma-2-2B	51.64
Qwen2.5-7B	70.76
Qwen2.5-3B	63.66

Table 5.11: Baseline accuracy of Vanilla models on the MMLU benchmark.

Model Combination	Accuracy (%)
One-block stitching	
Llama-3-8B – Gemma-2-2B	22.05
Qwen2.5-7B – Gemma-2-2B	23.49
Llama-3-8B – Qwen2.5-3B	25.99
Aligned one-block stitching	
Llama-3-8B – Gemma-2-2B	56.16
Qwen2.5-7B – Gemma-2-2B	19.37
Llama-3-8B – Qwen2.5-3B	34.74
Same tokenizer stitching	
Llama-3-8B – Gemma-2-2B	43.78
Qwen2.5-7B – Gemma-2-2B	26.61
Llama-3-8B – Qwen2.5-3B	50.57
Stitching in the SAE space	
Llama-3-8B – Gemma-2-2B	23.02
Qwen2.5-7B – Gemma-2-2B	22.91
Llama-3-8B – Qwen2.5-3B	22.94
Aligned stitching in the SAE space	
Llama-3-8B – Gemma-2-2B	28.30
Qwen2.5-7B – Gemma-2-2B	24.64
Llama-3-8B – Qwen2.5-3B	42.22
Same tokenizer stitching in the SAE space	
Llama-3-8B – Gemma-2-2B	22.68
Qwen2.5-7B – Gemma-2-2B	26.28
Llama-3-8B – Qwen2.5-3B	41.80

Table 5.12: MMLU Accuracy comparison across different stitching configurations.

Prompt	Model Completion
America is the	11th most expensive country in the world to live in, according to a new study.
Italy is the	11th largest economy in the world and the 6th largest in Europe.
Paris is the capital of	15,000,000 people, and it is the most free and the most}.
Bologna is a city	40 miles north of the city of <a href=”i 1:00:00.00:00:00.00

Table 5.13: Example generations produced by the hybrid model.

Prompt	Generated Token
If a round fruit is a tangerine, a 苹果 (apple) is an	*apple*
I use my 车 (car) to	*drive*
On the street I saw a 狗 (dog) chasing a	*dog*
She picked a 花 (flower) from the	*garden*
The farmer keeps his 牛 (cow) in the	*field*
At night the 天空 (sky) is full of	*the*
The 小猫 (cat) was sleeping on the	*couch*
In winter I wear a 外套 (coat) to stay	*safe*
If today is Monday, 明天 (tomorrow) will be	*the*

Table 5.14: Zero-shot cross-lingual generation examples. The first column contains a prompt with a Chinese keyword (with English translation for clarity), and second column reports the next token generated by the model.

Chapter 6

Future Work

Due to time constraints and computational limitations, this thesis could not exhaustively explore every combination of models and stitching modalities. Consequently, several promising avenues for future research emerge from the findings presented.

6.1 Scaling the Training Recipe

The primary limitation of the current work is the scale of the training process. The proposed methodology, in terms of dataset size, training steps, and batch size, is orders of magnitude smaller than standard LLM pre-training or fine-tuning protocols. Specifically, the maximum sequence length was limited, and the effective batch size (tokens per step) was significantly lower than the industry standard of 4 million+ tokens.

A larger batch size is crucial not just for speed, but for stabilizing the stochastic gradients during the alignment of two disparate high-dimensional spaces. Future iterations of this work should prioritize scaling up the training compute: increasing the sequence length to 4096 tokens and processing significantly more tokens per step would likely reduce the variance in the stitching layer's updates and yield more robust performance.

6.2 Investigating the ”Single Tokenizer” Paradox

One of the most counter-intuitive yet successful findings of this research is the viability of the ”Single Tokenizer” stitching modality. In this setup, the stitching layer learns to map the hidden states of Model A (Input) to Model B (Output) such that Model B produces logits compatible with Model A’s tokenizer, despite both model’s weights remaining frozen.

This implies the stitching layer is solving a complex optimization problem: it crafts an ”adversarial” input for Model B’s frozen layers that forces the output distribution to align with an alien vocabulary. To understand this mechanism, preliminary experiments were conducted to analyze the geometry of the hidden states.

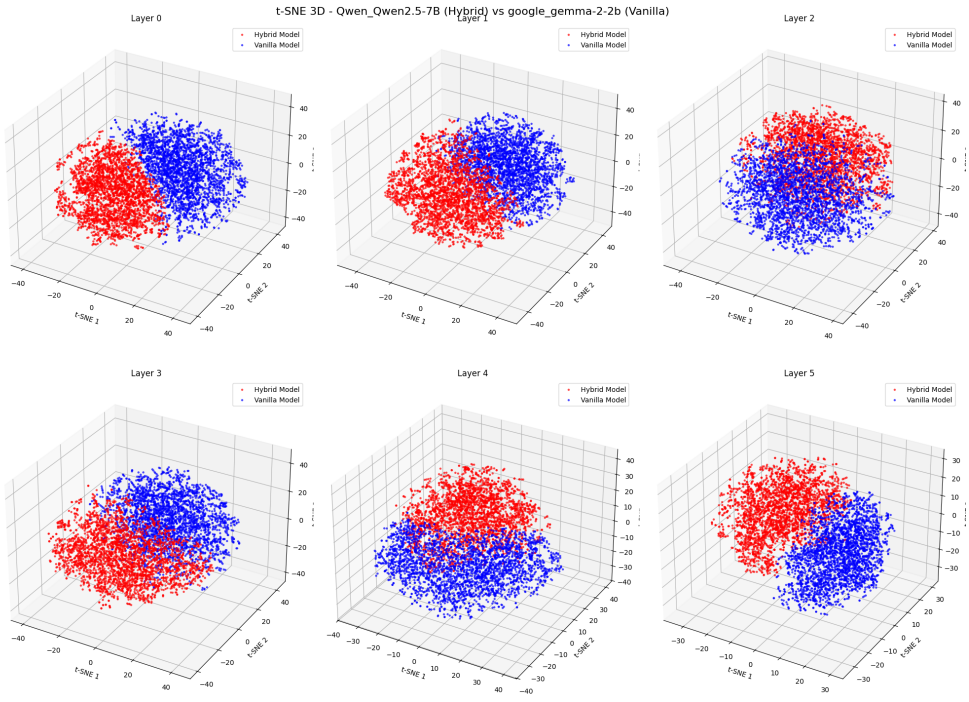

Figure 6.1: t-SNE visualization of hidden states in the target model. Red points represent the ”hybrid” states (output of stitching), while blue points represent the ”natural” states of the target model for the same tokens.

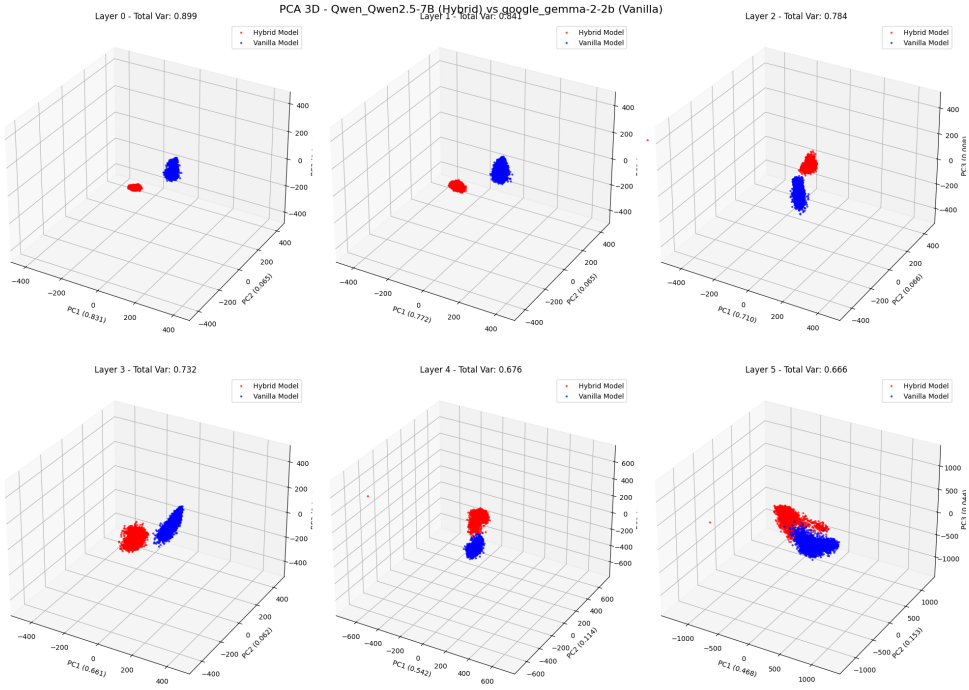

Figure 6.2: PCA projection of the hidden states. Note the distinct clustering between hybrid (stitched) and vanilla representations, despite encoding the same semantic token.

As visualized in Figures 6.1 and 6.2, the hidden states produced by the stitching layer (Hybrid) and the natural hidden states of the target model (Vanilla) for the same tokens do not overlap. Instead, they form distinct clusters.

- **Compact Subspace:** The PCA analysis reveals that the hybrid tokens occupy a much more compact subspace compared to the sparse distribution of the vanilla tokens.
- **Terminal Convergence:** Interestingly, while the intermediate representations are distinct, the clusters appear to ”collide” or converge towards the final layers.

This suggests that the stitching layer is not recreating the exact internal state of the target model. Rather, it has discovered a *functional equivalent* separate manifold in the high-dimensional space that, when processed by the target model’s frozen weights, triggers the correct unembedding vectors. Unraveling the precise geometry of this ”shortcut” mechanism is a crucial direction for future mechanistic interpretability studies. It challenges the assumption

that concept alignment requires precise state reconstruction, suggesting instead that the ”Semantic Hub” might be more flexible, allowing for multiple valid trajectories to the same semantic output.

6.3 Cross-Lingual and Multimodal Generalization

Finally, the zero-shot cross-lingual results presented in the previous chapter warrant a dedicated study. If the stitching layer can align Chinese concepts from Qwen to English concepts in Gemma without explicit supervision, this strongly supports the existence of language-agnostic universals in the Semantic Hub.

Future work should combine the dimensionality reduction analysis with cross-lingual inputs. By visualizing the trajectory of a concept like ”cat” (English) and ”mao” (Chinese) through the stitching layer, one could potentially observe the physical location of the ”universal concept” within the model’s geometry.

Conclusion

This thesis set out to validate the Semantic Hub Hypothesis through the lens of mechanistic interpretability, specifically targeting the challenge of inter-family model stitching. By attempting to bridge the latent spaces of distinct architectures, such as Llama, Gemma, and Qwen, this work sought to determine if different Large Language Models converge upon a universal, shared representation of concepts. The experimental results provide compelling empirical evidence that such a translation is practically achievable.

The findings demonstrate that a linear transformation is sufficient, when employing ad hoc trained sparse autoencoders, to translate the ”thinking process” of one model into the hidden state of another. The success of the *Llama-3* to *Qwen-2.5* stitching configuration on benchmarks like HellaSwag indicates that semantic reasoning capabilities can be preserved across architectural boundaries. Furthermore, the counter-intuitive viability of the ”Single Tokenizer” setup suggests that high-level semantic alignment is robust enough to overcome low-level vocabulary mismatches, effectively forcing a model to operate within an alien token space.

However, this work must be contextualized as a preliminary investigation. The constraints imposed by limited computational resources and time necessitated a focused scope, leaving several critical variables unexplored. Specifically, the optimal placement of the stitching layer remains an open question: I restricted my experiments to a single depth (70-80%), but a comprehensive sweep might reveal more efficient translation points. Similarly, the training recipe was scaled down for domestic hardware, and the potential gains from extensive hyperparameter tuning and larger-scale training on temporally aligned datasets have yet to be fully realized.

Despite these limitations, the core conclusion remains positive: inter-family model stitching is possible. The barriers between different model families

are not insurmountable walls of incompatible logic, but rather differences in dialect that can be translated. This thesis lays the groundwork for a unified understanding of Artificial Intelligence, moving away from viewing models as isolated black boxes. While the bridge built here is a prototype, it proves that the shores of different model families are connected by the same semantic ground. Future research, leveraging the directions proposed herein, is essential to fully map this terrain and complete the picture of the universal semantic hub.

Bibliography

[1] D. Amodei. The urgency of interpretability. April 2025. URL: www.darioamodei.com/post/the-urgency-of-interpretability (visited on 09/02/2025).

[2] F. Bello, A. Das, F. Zeng, F. Yin, and L. Leqi. Linear representation transferability hypothesis: leveraging small models to steer large models, 2025. arXiv: 2506.00653 [cs.LG]. URL: https://arxiv.org/abs/2506.00653.

[3] J. Camacho-Collados and M. T. Pilehvar. From word to sense embeddings: a survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788, 2018.

[4] A. Chen, J. Merullo, A. Stolfo, and E. Pavlick. Transferring features across language models with model stitching, 2025. arXiv: 2506.06609 [cs.CL]. URL: https://arxiv.org/abs/2506.06609.

[5] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, et al. The Pile: an 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[6] A. Gokaslan, V. Cohen, E. Pavlick, and S. Tellex. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[7] Google. Gemini. 2025. URL: https://www.google.com/gemini (visited on 09/02/2025). AI-powered large language model.

[8] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL: https://openreview.net/forum?id=d7KBjmI3GmQ.

[9] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu. A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, January 2025. ISSN: 1558-2868. DOI: 10.1145/3703155. URL: http://dx.doi.org/10.1145/3703155.

[10] M. Kosinski. What is black box ai and how does it work. October 29, 2024. URL: https://www.ibm.com/think/topics/black-box-ai (visited on 09/02/2025).

[11] S. Moore. Google search is dead. 2025. URL: https://www.trendmill.com/p/google-search-is-dead. Accessed: 2025-09-02.

[12] C. Olah. Mechanistic interpretability, variables, and the importance of interpretable bases. 2022. URL: www.transformer-circuits.pub/2022/mech-interp-essay (visited on 09/02/2025).

[13] N. F. Oozeer, D. Nathawani, N. Prakash, M. Lan, A. HARRASSE, and A. Abdullah. Activation space interventions can be transferred between large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL: https://openreview.net/forum?id=HXOicJsmMQ.

[14] OpenAI. Chatgpt. 2025. URL: https://chat.openai.com/ (visited on 09/02/2025). Large language model, version GPT-5.

[15] OpenAI. Introducing gpt-5. 2025. URL: https://openai.com/index/introducing-gpt-5/. Accessed: 2025-09-02.

[16] Perplexity AI. Perplexity ai. 2025. URL: https://www.perplexity.ai (visited on 09/02/2025). AI-powered search and chat assistant.

[17] Redazione. Chatgpt a scuola, valditara: ”intelligenza artificiale offre straordinarie opportunità, ma il ruolo del docente è insostituibile”. 2025. URL: https://www.orizzontescuola.it/chatgpt-a-scuola-valditara-intelligenza-artificiale-offre-straordinarie-opportunita-ma-il-ruolo-del-docente-e-insostituibile/. Accessed: 2025-09-02.

[18] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 1715–1725, 2016.

[19] D. Shu, X. Wu, H. Zhao, D. Rai, Z. Yao, N. Liu, and M. Du. A survey on sparse autoencoders: interpreting the internal mechanisms of large language models. In C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 1690–1712, Suzhou, China. Association for Computational Linguistics, November 2025. ISBN: 979-8-89176-335-7. DOI: 10.18653/v1/2025.findings-emnlp.89.

[20] H. Thasarathan, J. Forsyth, T. Fel, M. Kowal, and K. Derpanis. Universal sparse autoencoders: interpretable cross-model concept alignment, 2025. arXiv: 2502.03714 [cs.CV]. URL: https://arxiv.org/abs/2502.03714.

[21] A. M. Turner, L. Thiergart, G. Leech, D. Udell, J. J. Vazquez, U. Mini, and M. MacDiarmid. Steering language models with activation engineering, 2024. arXiv: 2308.10248 [cs.CL]. URL: https://arxiv.org/abs/2308.10248.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[23] T. Wang, X. Jiao, Y. Zhu, Z. Chen, Y. He, X. Chu, J. Gao, Y. Wang, and L. Ma. Adaptive activation steering: a tuning-free llm truthfulness improvement method for diverse hallucinations categories. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pages 2562–2578, Sydney NSW, Australia. Association for Computing Machinery, 2025. ISBN: 9798400712746. DOI: 10.1145/3696410.3714640. URL: https://doi.org/10.1145/3696410.3714640.

[24] Z. Wu, X. Yu, D. Yogatama, J. Lu, and Y. Kim. The semantic hub hypothesis: language models share semantic representations across languages and modalities. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu,

editors, *International Conference on Learning Representations*, volume 2025, pages 53705–53723, 2025.

[25] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

Acknowledgements

Volevo ritagliare questo spazio per ringraziare le persone con le quali ho condiviso, direttamente o indirettamente, quello che è probabilmente l' ultimo passo del mio percorso universitario. In primis desidero ringraziare la mia famiglia, senza la quale non avrei nemmeno avuto l' opportunità di iniziare questo percorso e che mi ha sempre fornito tutto il supporto necessario. Ci tengo inoltre a ringraziare Andrea Capitani e tutti i colleghi di Expert.ai, che mi hanno supportato durante tutto il lavoro di tirocinio, aiutandomi a interpretare i risultati e a discutere insieme i possibili passi successivi, che è culminato nella scrittura di questa tesi. Ringrazio inoltre il professore Paolo Torroni e il dottor Federico Ruggeri, che mi hanno fornito un grande aiuto nella stesura di questo lavoro. Infine, ma non per importanza, desidero ringraziare i miei amici, che mi hanno aiutato a superare alcuni momenti non semplici durante questo percorso, e i miei colleghi universitari, con i quali ho instaurato un bellissimo rapporto e con cui ho condiviso, direttamente o indirettamente, le difficoltà e le soddisfazioni di questo cammino.