Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Interpreting the Reversal Curse of LLMs

Master Thesis

Giovan Battista Landolina

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Sam Dauncey
Prof. Dr. Roger Wattenhofer

November 1, 2025

Acknowledgements

I would like to thank my supervisors, Sam Dauncey and Prof. Roger Wattenhofer, for their guidance and for allowing me the freedom to pursue my ideas independently.

I am deeply grateful to my family, my mum, dad, and sister, as well as to my friends and colleagues, for their constant support, encouragement, and understanding throughout this work and during the challenges of student life.
Parts of this thesis were refined using OpenAI's ChatGPT language model.

Abstract

This thesis investigates the origins and mechanisms of the *reversal curse* in modern autoregressive Large Language Models.
Building on previous work showing that such models can overcome the reversal curse in controlled synthetic settings, we analyze *how* this occurs using methods from mechanistic interpretability, identifying the internal circuits responsible for reversal learning.
We then examine the gap between synthetic and realistic scenarios, clarifying why the curse persists in naturalistic conditions and what this reveals about the inductive biases of LLMs.
Finally, we discuss broader implications for the generalization and reasoning capabilities of language models that emerge from this analysis.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1
1.1 Motivation . . . 1
1.2 Problem Statement . . . 1
1.2.1 Definition of the Reversal Curse . . . 2

2 Related Work 3
2.1 Introduction to Transformers and Large Language Models . . . 3
2.1.1 Deep Learning . . . 3
2.1.2 Feedforward Neural Networks . . . 5
2.1.3 Training Feedforward Networks: Loss Functions and Gradient Descent . . . 7
2.1.4 Transformers . . . 9
2.1.5 Language Models . . . 12
2.2 Reversal Curse . . . 13
2.2.1 The NTP Failure Hypothesis . . . 14
2.2.2 Data Augmentation Strategies . . . 15
2.2.3 Alternative Optimization Objectives . . . 16
2.2.4 Shortcut Learning . . . 19
2.2.5 Input-Output Mismatch Hypothesis . . . 19
2.3 Interpretability techniques . . . 21
2.3.1 Causal Tracing . . . 21
2.3.2 Layerwise Decoding (Logit/Tuned Lens) to Measure Emergent Accuracy 23
2.3.3 Zero-Ablation of Submodules to Test Necessity . . . 24

3 Results 25
3.1 ROME and the Reversal Curse . . . 25
3.2 Indirect Training . . . 29
3.2.1 Introduction and Initial results . . . 29
3.2.2 Bridge Entity Hypothesis . . . 31
3.3 Relational Embedding Models . . . 34
3.3.1 TransE . . . 34
3.3.2 RotatE . . . 34
3.3.3 ComplEx . . . 34
3.3.4 SimplE . . . 35
3.3.5 Training Procedure . . . 35
3.3.6 Conceptual and Empirical Comparison . . . 35
3.4 Grokking the reversal curse . . . 36
3.4.1 Experimental Setup . . . 36
3.4.2 Static Mechanistic Interpretability . . . 37
3.5 Symmetry and the Emergence of Generalization . . . 43
3.5.1 Symmetry in the Attention Operator and Its Role in Generalization . . . 44
3.5.2 Embedding Geometry and Relational Structure . . . 46

3.5.3 Bilinear Training 48
3.5.4 Gap between synthetic and realistic settings 52

4 Conclusion **53**

Chapter 1

Introduction

1.1 Motivation

Machine learning methods and applications have recently left research laboratories and entered the everyday life of millions of people. Among their many areas of success, natural language processing has been particularly transformative: Large Language Models (LLMs) have demonstrated remarkable abilities that few could have anticipated before the introduction of the transformer architecture.

Tools such as ChatGPT are significantly faster and more interactive than traditional web search engines. They can summarize long texts in a fraction of a second, enabling users to absorb large amounts of information quickly, and they understand language with striking fluency, providing guidance for writing, summarization, and other complex linguistic tasks.

However, the surprising power of these architectures has led us to demand even more from them. Many now treat LLMs less as simple language models and more as general-purpose oracles. As a result, both academia and industry are focusing considerable effort on addressing the problem of hallucination in these models. This concern arises because users now expect LLMs to produce error-free code, rigorous proofs, and reliable answers to complex questions, tasks that require genuine reasoning rather than mere pattern recognition.

Despite their impressive performance, the true generalization and reasoning capabilities of LLMs remain the subject of active debate (Shojaee et al. [2025] Zhang [2025]).
A common view is that these models largely rely on memorization of massive text corpora and excel at retrieving and composing plausible continuations, which may appear correct but are not the product of actual reasoning. Phenomena such as hallucinations suggest that we are still far from developing systems capable of human-level deduction and abstract understanding.

Indeed, LLMs exhibit well-documented generalization failures, such as the two-hop curse Balesni et al. [2024] and the **reversal curse** Berglund et al. [2023], the latter being the central focus of this thesis. We argue that studying such failure cases, and understanding the conditions under which they can be mitigated, is crucial both for designing better models and for clarifying the current limitations of AI systems.

1.2 Problem Statement

This work aims to provide a comprehensive explanation of why Large Language Models (LLMs) suffer from the reversal curse, one of the fundamental limitations observed in these architectures. As discussed previously, studying such failure cases can be extremely insightful and can lead to a deeper understanding of the internal mechanisms and limitations of LLMs. Despite being the subject of several studies since its discovery, the reversal curse has never been fully explained in

detail. Most existing research has focused on mitigating or patching the problem, rather than investigating its underlying causes.

In this work, we address the following questions:

1. Why does the reversal curse arise in LLMs? Do all LLMs suffer from it?
2. Under which conditions can these models learn to overcome the reversal curse?

We now proceed to formally define the reversal curse in the context of LLMs.

1.2.1 Definition of the Reversal Curse

The *reversal curse* refers to the inability of autoregressive language models to generalize bidirectional relations expressed as subject–relation–object triples. Formally, given a factual association represented as a triplet

$$t = (s, r, o),$$

where s denotes the subject, r the relation, and o the object, a model trained to predict o from the prompt (s, r) often fails to predict s from the reversed prompt (o, r^{-1}), even though both correspond to logically equivalent facts.

In other words, the model can learn that

"Paris is the capital of France"

corresponds to the association (Paris, capital_of, France), but it does **not** automatically infer that

"France has Paris as its capital"

should also hold.

This phenomenon highlights an asymmetry in the way autoregressive models encode and retrieve factual knowledge. Rather than forming structured, relational representations, they tend to memorize unidirectional statistical mappings between tokens. The reversal curse thus serves as a concrete example of the broader limitation of LLMs in performing *relational reasoning* and *structured generalization*.

Chapter 2

Related Work

2.1 Introduction to Transformers and Large Language Models

In this section, we provide a brief recap and introduction to deep learning, particularly focusing on the transformer architecture, which is the foundational basis of many modern large language models.
Detailed knowledge of how these architectures work will be essential to understand our later experiments, as well as gaining a better grasp of the problem in general.
Readers who are already familiar with these topics may proceed directly to the next section.

2.1.1 Deep Learning

Deep learning is a branch of computer science, and more specifically, a subfield of machine learning. Like machine learning in general, it aims to design intelligent systems capable of imitating human behavior and solving complex tasks.

The main difference between these methods and traditional computer algorithms lies in how they acquire problem-solving capabilities. In classical programming, an algorithm is explicitly designed to follow a precise, human-defined set of instructions. In contrast, machine learning systems undergo a training phase, during which they are not told the exact procedure to solve a task. Instead, they learn the underlying algorithm automatically by detecting patterns and regularities within structured data.

Deep learning is different from other classical machine learning approaches for its use of complex, multilayer architectures, often referred as neural networks.
Learning a single mapping from the input to the output space can be extremely challenging, and for this reason, deep learning architectures usually employ several layers, breaking down a complex problem in several, slightly easier subtasks.
These models are often hierarchical, meaning that the patterns they learn gradually increase in complexity, from the shallowest to the deepest layers. Each layer builds upon the patterns extracted by the previous ones and combines them to create higher-level features.

To illustrate an example, we focus on a computer vision model that, given raw images of cats and dogs, is able to classify them accurately.
The first layers of these models are usually responsible for extracting low-level features, like edges, textures, color gradients, etc.
This information is then used by the intermediate layers to form higher level patterns, for example eyes or ears.
Finallly, the deepest layers are responsible for combining the high-level features to build a successful classifier (for instance: round ears and a broad snout $\rightarrow$ dog, pointed ears and slit pupils $\rightarrow$ cat.)

These architectures are invaluable, since they allow us to solve complex tasks automatically, without hardcoding the rules or requiring humans to manually design the patterns or classification criteria.
It is important to note, however, that these models do not always find clear, human interpretable features (like ears or fur), but might rely on more complicated patterns that humans would not easily design, being often complex and counterintuitive.
The previous example was introduced to give the reader an intuitive understanding of how deep learning architectures work and how they achieve such remarkable results.
Neural networks are, indeed, often treated like black-box architectures and deemed too difficult to interpret or debug.
This comes with an inherent trade-off between interpretability and complexity (and therefore often power) of such models.
The ability to derive features that humans would rarely conceive is one of the key reasons for their success over classical machine learning methods.

Recently, large language models (LLMs), deep learning architectures designed for natural language, have achieved remarkable success but have also introduced new challenges related to privacy, security, and reliability. As a result, research on the interpretability and transparency of these models has gained significant momentum, providing valuable insights for improving model alignment, safety, and trustworthiness.

On the other hand, although these methods have proven highly successful across a wide range of tasks, from natural language processing and computer vision to protein structure modeling and other scientific applications, they also come with their own set of limitations.
In many cases, indeed, simpler machine learning approaches are preferable for several reasons.
As previously discussed, neural networks are considerably harder to understand, compared to models such as random forests or logistic regression.
Other major limitations include the massive amount of data and the dedicated hardware they require during the training phase.
Many models need several days of training on multiple GPUs or TPUs, making them highly expensive in terms of resources and energy usage.

Despite these challenges, deep learning models appear to be highly promising, and it is worth studying why they are so successful and effective.

From a theoretical perspective, their success relies on two fundamental principles underlying nearly all neural network architectures:

The **Universal Approximation Theorem**, which explains the expressive power of neural networks, why they have the capacity, in theory, of modeling arbitrarily complex relationships between inputs and outputs.

Gradient-based optimization, and in particular the combination of gradient descent and backpropagation, which describes how networks actually learn these relationships from data.

Together, these ideas form the theoretical and practical backbone of modern deep learning.
Before diving into the details, we formally define Feedforward Neural Networks, the most general neural networks and the ones that are usually used to study the theoretical capabilities and limitations of these models.

2.1.2 Feedforward Neural Networks

A *feedforward neural network*, the simplest deep learning architecture, is comprised of several layers, each containing a set of neurons, that are fully connected to every other neuron in the adjacent layers. The strength of the connections is determined by a set of weights, which the model learns during the optimization process.
Information in the model flows in only one direction, from the input to the output layer, and never in the opposite direction.
The architecture is loosely inspired by the human (or animal) brain, and shares many similarities. Just as the brain strenghtens its most useful pathsways, the training phase adjusts the network's weights to reinforce the most useful connections.
At the same time, just like a biological brain "fires" when the electrical signal exceeds a certain threshold, an artificial neuron becomes activated if the combination of its inputs surpasses a given activation threshold (this is especially true with ReLU as activation function).

Mathematical Formulation

Let $x_j^{(l-1)}$ denote the activation (output) of the j-th neuron in layer $l-1$, and $x_i^{(l)}$ the activation of the i-th neuron in the next layer l. Each connection between two neurons has an associated *weight* w_{ij}, and each neuron has an additional *bias* term b_i. The computation performed by neuron i in layer l can be written as:

$$x_i^{(l)} = \sigma\left(\sum_j w_{ij}\, x_j^{(l-1)} + b_i\right), \tag{2.1}$$

where $\sigma(\cdot)$ is a *nonlinear activation function*, which determines whether and to what extent the neuron "fires." This nonlinearity is essential: if all layers used only linear operations (i.e., matrix multiplications and additions), the entire network could be reduced to a single linear transformation, regardless of its depth. Nonlinear activation functions therefore allow neural networks to approximate complex, nonlinear mappings between inputs and outputs.

Common Activation Functions

Different choices of activation function σ lead to different learning behaviors and numerical properties. Below are some of the most widely used ones:

Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

Historically one of the first activation functions, the sigmoid maps inputs to the range $(0, 1)$, mimicking biological neuron firing rates. However, it suffers from *vanishing gradients* for large $|x|$, which slows or prevents training in deep networks.

Hyperbolic tangent (tanh)

$$\sigma(x) = \tanh(x) \tag{2.3}$$

Similar in shape to the sigmoid but centered around zero, the tanh function produces outputs in $(-1, 1)$, often resulting in faster convergence. Nonetheless, it still suffers from gradient saturation for large values.

Rectified Linear Unit (ReLU)

$$\sigma(x) = \max(0, x) \tag{2.4}$$

ReLU is now the most commonly used activation in deep networks. It is computationally efficient, helps mitigate the vanishing gradient problem, and enables sparse activations (many neurons output zero). A drawback, however, is the *dying ReLU* problem: neurons that output only zeros may never recover during training.

Leaky ReLU and Variants To address the dying ReLU issue, the Leaky ReLU allows a small negative slope:

$$\sigma(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{2.5}$$

where α is a small constant (e.g., 0.01), ensuring that gradients never vanish completely.

Gaussian Error Linear Unit (GELU) The GELU activation, introduced by **?**, is defined as:

$$\sigma(x) = x\,\Phi(x) = x \cdot \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \tag{2.6}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. Intuitively, GELU weights inputs by their probability of being positive under a Gaussian distribution, providing a smooth alternative to ReLU. It has become the default activation in Transformer architectures such as GPT and BERT, as it combines the benefits of ReLU's nonlinearity with smoother gradient flow.

Softmax Used in the output layer for classification tasks, the softmax function converts a vector of raw scores z_i into a probability distribution:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.7}$$

The choice of activation function depends on the specific task and network depth. Modern architectures often use ReLU or GELU in intermediate layers and softmax for outputs, while sigmoid or tanh may still appear in specialized applications such as recurrent networks or probabilistic models.

The Universal Approximation Theorem Cybenko [1989] states that a feedforward neural network with at least one hidden layer and a sufficient number of neurons can approximate any continuous function on a compact domain, given an appropriate choice of parameters and activation functions.

In simple terms, this means that, regardless of the complexity of a task, be it image recognition, language modeling, or physical simulation, a sufficiently large neural network has the potential to represent the underlying mapping from inputs to outputs. This theoretical result does not guarantee that the network will find the correct parameters or that it will generalize well to unseen data, but it establishes that neural networks are expressive enough to model virtually any functional relationship.

The theorem provides the conceptual foundation for deep learning: depth and nonlinearity give neural networks the flexibility to approximate high-dimensional, nonlinear mappings that would be intractable for traditional models.

2.1.3 Training Feedforward Networks: Loss Functions and Gradient Descent

The training procedure of a neural network consists in finding a set of parameters (weights) that minimize a given measure of error, known as the *loss function*.
This is generally achieved through iterative optimization using gradient-based methods.

In this section, we introduce the concepts of loss functions, gradient descent, and the backpropagation algorithm, which together enable efficient end-to-end learning in deep neural networks.

Loss Functions

A loss function $\mathcal{L}(\hat{y}, y)$ measures the discrepancy between the model's predicted output $\hat{y}$ and the true target value y. During training, the network aims to minimize the average loss over all training examples:

$$\mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\hat{y}^{(n)}, y^{(n)}), \tag{2.8}$$

where N denotes the number of samples in the training set.

The choice of loss function depends on the task:

- For **regression** problems, a common choice is the Mean Squared Error (MSE):

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}^{(n)} - y^{(n)})^2. \tag{2.9}$$

- For **classification** problems, the Cross-Entropy loss is typically used:

$$\mathcal{L}_{\text{CE}} = -\sum_{c=1}^{C} y_c \log \hat{y}_c, \tag{2.10}$$

  where C is the number of classes, y_c is the true label (often one-hot encoded), and $\hat{y}_c$ is the predicted probability for class c.

Minimizing this loss function is considered to be equivalent to maximizing the likelihood of the model parameters, given the data.
From an information-theoretical perspective, it is encouraged because it is equivalent to minimizing the Kullback-Leibler divergence between the true data distribution and the model's predicted distribution. Practically speaking, this loss encourages the model to find a distribution that aligns with the data observed in the training set.

Gradient Descent Optimization

Neural networks are typically trained using variants of *gradient descent*, an optimization algorithm that iteratively updates the model parameters in the direction that reduces the loss. Given a parameter θ (representing a weight or bias), its update rule at iteration t is:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial \mathcal{L}}{\partial \theta_t}, \tag{2.11}$$

where $\eta > 0$ is the *learning rate*, a hyperparameter controlling the step size of each update.

The gradient $\frac{\partial \mathcal{L}}{\partial \theta_t}$ measures how much the loss would change with respect to a small variation in θ_t. Intuitively, the algorithm moves each parameter in the direction opposite to the gradient of the loss, i.e., toward regions of lower loss.

In practice, computing the gradient over the entire training set can be computationally expensive. To address this, deep learning employs *stochastic* or *mini-batch* gradient descent, where gradients are estimated from small subsets (batches) of data. This not only reduces computational cost but also introduces a beneficial stochasticity that can help escape local minima.

Common extensions include adaptive optimization algorithms such as **Adam** , **RMSProp**, and **Adagrad**, which dynamically adjust learning rates for each parameter based on gradient statistics, often leading to faster and more stable convergence.

Backpropagation

While gradient descent specifies how parameters are updated, the efficient computation of gradients for networks with many layers is made possible by the *backpropagation* algorithm. Backpropagation applies the chain rule of calculus to propagate error signals backward through the layers of the network, layer by layer, enabling the computation of partial derivatives with respect to all parameters.

Consider the neuron equation:

$$x_i^{(l)} = \sigma\left(\sum_j w_{ij}\, x_j^{(l-1)} + b_i\right). \tag{2.12}$$

During training, we compute how much the loss changes with respect to each parameter:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial x_i^{(l)}} \cdot \frac{\partial x_i^{(l)}}{\partial w_{ij}}. \tag{2.13}$$

The term $\frac{\partial \mathcal{L}}{\partial x_i^{(l)}}$ is recursively obtained from the subsequent layer via:

$$\frac{\partial \mathcal{L}}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \mathcal{L}}{\partial x_i^{(l)}} \cdot \frac{\partial x_i^{(l)}}{\partial x_j^{(l-1)}}. \tag{2.14}$$

This recursive computation proceeds from the output layer back to the input layer—hence the name *backpropagation*. Modern deep learning frameworks (such as PyTorch or TensorFlow) implement this process automatically through *automatic differentiation*, allowing the user to focus on model design rather than gradient derivation.

Training Summary

In summary, training a neural network involves the following iterative steps:

1. Perform a **forward pass** through the network to compute predictions.
2. Evaluate the **loss function** to quantify prediction error.
3. Compute gradients via **backpropagation**.
4. Update parameters using **gradient descent**.

Through repeated application of these steps over many epochs, the network gradually refines its internal parameters, reducing the loss and improving predictive performance. This training pipeline forms the core of nearly all deep learning systems, from simple feedforward networks to complex architectures such as convolutional and transformer-based models.

2.1.4 Transformers

The *Transformer* architecture, introduced by Vaswani et al. [2017], represents one of the most important breakthroughs in modern deep learning.
Unlike earlier NLP architectures, this model does not use any recurrent neural network (RNNs) or any other form of sequential processing, and instead relies completely on the *self-attention* mechanism.
This design allows the model to be highly parallel and, at the same time, facilitates capturing long-range dependencies easily, making it one of the most influential architectures in contemporary deep learning.

Self-Attention To understand the Transformer architecture, it is essential to first introduce its core component: the **self-attention mechanism**.

As discussed earlier, in a feedforward neural network each neuron's activation is computed as a weighted combination of all activations from the previous layer. These weights are fixed for a given model and do not depend on the specific input being processed. However, in many tasks, especially in natural language processing, it is desirable for these weights to be *dynamic*, adapting to the context of the input sequence.

For instance, when predicting the next word in a sentence, we might want to focus on different parts of the text depending on its meaning. In the phrase "The cat sat on the mat because it was soft," the word *it* should attend primarily to *the mat*, while in "The cat sat on the mat because it was tired," the same word *it* refers instead to *the cat*. A fixed set of weights cannot capture such variable dependencies effectively.

The self-attention mechanism is designed to model precisely such cases. It works by dynamically determining how much *attention* each token should pay to every other previous token in the sequence.
In other words, the model learns to identify context-dependent relationships within a sequence and to capture the associations between its constituent elements.

This concept is not limited to language: similar mechanisms have proven highly effective in computer vision, genomics, and various multimodal and scientific applications, wherever relationships among elements depend on their content rather than their fixed position or order.

Mathematical Formulation of Self-Attention Formally, consider an input sequence represented as a matrix $X \in \mathbb{R}^{T \times d_{\text{model}}}$, where T indicates the sequence length and d_{model} is the dimensionality of each token embedding. Each token in the sequence is linearly projected into three different vector spaces, producing the *queries*, *keys*, and *values*:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \tag{2.15}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learnable parameter matrices and d_k is the dimensionality of the query and key vectors.

The core idea of self-attention is to compute a similarity score between each query and all keys in the sequence. Intuitively, if token i attends strongly to token j, the query vector associated with token i should be well aligned with the key vector corresponding to token j.
For instance, consider the sentence "Paris is the capital of France". The model will likely assign a high attention weight to the couple (Paris, France) since the query vector representing the concept "capital of ..." will be similar, semantically, to the key vector of Paris. This operation is performed for every pair of tokens in the sequence (although, in autoregressive models such as GPT, each token is only allowed to attend to previous tokens). This results in a matrix of

attention scores $S \in \mathbb{R}^{T\times T}$, where each element S_{ij} is a measure of how much the token at position i should attend to the token at position j:

$$S = \frac{QK^\top}{\sqrt{d_k}}. \tag{2.16}$$

The scaling factor $\frac{1}{\sqrt{d_k}}$ prevents the dot products from growing too large in magnitude. Without this normalization, the dot products would saturate, resulting in very small gradients and consequently hindering effective training.

These raw attention scores are then normalized using the *softmax* function along each row, producing a matrix $A \in \mathbb{R}^{T\times T}$ of non-negative weights that sum to one:

$$A_{ij} = \frac{\exp(S_{ij})}{\sum_{j'} \exp\big(S_{ij'}\big)}. \tag{2.17}$$

The resulting attention weights define, for each token i, a probability distribution over all other tokens in the sequence. Hence, the output of the attention mechanism is a *convex combination* of the value vectors, where each token representation is updated according to the weighted contributions of all tokens:

$$Z = AV, \tag{2.18}$$

where $Z \in \mathbb{R}^{T\times d_k}$ represents the output of the self-attention operation before projection.

To allow the transformer to model different interactions at the same time, in many applications, several independent attention "heads" are used in parallel. This is known as *multi-head attention*. Each head computes its own set of Q, K, V projections, applies self-attention independently, and the results are concatenated and linearly transformed by a learned output projection matrix $W_O \in \mathbb{R}^{(h\cdot d_k)\times d_{\text{model}}}$:

$$\text{MultiHead}(X) = \text{Concat}(Z_1, Z_2, \ldots, Z_h)W_O. \tag{2.19}$$

This multi-head structure allows the model to attend to different types of relationships simultaneously (e.g., syntactic vs. semantic dependencies in text).

The Transformer Block After having introduced the self-attention mechanism, we can now define the fundamental building unit of the transformer model.

Each Transformer block consists of two main submodules:

1. a **Multi-Head Self-Attention** component, responsible for computing contextual associations between tokens
2. a **FeedForward Neural Network**, applied independently to each token of the sequence in order to further refine each representation and enrich its semantic representation.

Both subcomponents are wrapped with *residual connections* and followed by *layer normalization*, forming the following computation pattern:

$$X' = \text{LayerNorm}\left(X + \text{MultiHeadAttention}(X)\right), \tag{2.20}$$

$$Y = \text{LayerNorm}\left(X' + \text{FeedForward}(X')\right). \tag{2.21}$$

The residual connection encourages the model to learn small, incremental transformations of the input, instead of completely new representations. This makes the learning process more stable and efficient. Moreover, skip connections are also highly beneficial for mitigating the *vanishing gradient problem*, facilitating gradient flow through the network.
Another important subcomponent that we need to mention is layer normalization, which ensures consistent scaling across the network and further enhances convergence.

Feedforward Sublayer The feedforward sublayer (FFN) is typically implemented as a two-layer fully connected network applied independently to each token position:

$$\text{FeedForward}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2, \tag{2.22}$$

where $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ are learned parameter matrices, and d_{ff} is usually several times larger than d_{model} (commonly $d_{\text{ff}} = 4\, d_{\text{model}}$). The *Gaussian Error Linear Unit (GELU)* activation introduces smooth nonlinearity, allowing for expressive yet stable transformations of each token embedding.

While the self-attention sublayer captures dependencies *across* different tokens, the feedforward sublayer performs nonlinear transformations *within* each token's embedding space, enriching the learned representation dimensionally.

Stacking Transformer Blocks The transformer architecture consists of multiple Transformer blocks stacked sequentially. The input sequence is handled by a tokenizer, and then mapped to a high dimensional space through an embedding layer. After that, each token representation is subsequently refined across layers progressively integrating information from other tokens and from the model itself.
As mentioned earlier, stacking multiple layers allows the model to learn complex end-to-end transformations. By decomposing the problem across depth, the network enables each layer to learn simpler, incremental mappings, leading to a gradual transformation of the input representations.

For instance, the lower layers of a large language model (LLM) tend to capture syntactic and structural features, while higher layers, building upon these, encode richer semantic and contextual information. Small models such as GPT-2 typically consist of around 12 layers, whereas larger models like GPT-3 can include hundreds.

This modular and scalable design, combining self-attention, feedforward processing, normalization, and residual pathways, is what makes the Transformer architecture both highly expressive and computationally efficient, forming the foundation for nearly all modern large language models.

Layer Normalization In addition to attention and feedforward operations, each Transformer layer applies a technique called *layer normalization* Ba et al. [2016], which improves numerical stability and accelerates convergence. Given an input vector $x \in \mathbb{R}^{d_{\text{model}}}$, layer normalization normalizes its components to have zero mean and unit variance:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \gamma + \beta, \tag{2.23}$$

where μ and σ^2 are the mean and variance of the components of x, ϵ is a small constant for numerical stability, and $\gamma, \beta \in \mathbb{R}^{d_{\text{model}}}$ are learnable scaling and shifting parameters.

Layer normalization helps maintain consistent activation statistics throughout training, mitigating the problem of internal covariate shift and making optimization more stable. In practice, it is applied after both the attention and feedforward sublayers, each combined with a residual connection, forming the following pattern within a Transformer block:

$$X' = \text{LayerNorm}(X + \text{SelfAttention}(X)), \quad Y = \text{LayerNorm}(X' + \text{FeedForward}(X')). \tag{2.24}$$

This residual-normalized structure allows the model to preserve information across layers while refining it through attention and non-linear transformations, contributing to the remarkable training stability and expressiveness of Transformer architectures.

2.1.5 Language Models

Large Language Models (LLMs) have recently emerged as the most prominent and successful architectures in the field of natural language processing (NLP).
They are able to generate coherent text, to address the user's questions correctly and to get involved into complex reasoning.
For these reasons, they are heavily studied and considered in both academia and industry.
As we have explained, these models build on the transformer architecture, and exploit the ability of such architectures to learn statistical associations and regularities of language from large amounts of data.
We now classify LLMs in different types, based primarely on the objective they have followed during training. In particular, we will focus on the two main paradigms used today: *masked language modeling* and *next-token prediction*.

Masked Language Modeling (MLM) Masked language models, represented by architectures such as **BERT** Devlin et al. [2019], are trained using the *masked language modeling* objective. During training, a random subset of the input tokens is replaced by a special `[MASK]` token, and the model is asked to predict the identity of these masked tokens based on the surrounding, unmasked context.

Formally, given an input sequence of tokens $(x_1, x_2, \ldots, x_T)$, a fraction (typically 15%) of positions are selected for masking. For each masked position i, the model outputs a probability distribution over the entire vocabulary V:

$$p_\theta(x_i \mid x_1, \ldots, x_{i-1}, \texttt{[MASK]}, x_{i+1}, \ldots, x_T). \tag{2.25}$$

The model is then trained to minimize the cross-entropy loss between this predicted distribution and the true token at the masked position:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in M} \log p_\theta(x_i). \tag{2.26}$$

Forcing the model to predict the missing tokens from the adjacent context (in both directions, left and right) allows it to build a solid understanding of language and the relations between entities in text.
For this reason, BERT is often used as a foundational model for language understanding, providing rich representation that can be leveraged for downstream tasks, such as sentiment analysis, semantic similarity or question answering.

Next-Token Prediction (Autoregressive Modeling) In contrast, models such as **GPT** Radford et al. [2018] are trained using the **next-token prediction** (or autoregressive) objective. Here, the model is exposed to a sequence of tokens and learns to predict the next token given all previous ones. Formally, the model estimates the probability distribution over sequences by factorizing it as:

$$p_\theta(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t}), \tag{2.27}$$

and the training loss corresponds to the negative log-likelihood:

$$\mathcal{L}_{\text{NTP}} = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}). \tag{2.28}$$

This formulation enforces a unidirectional,or causal bias on the model, which is trained to predict the next token based solely on the preceding ones.
This constraint is implemented using a triangular attention mask that effectively ensures all attention weights from any given token to any subsequent token are set to zero.
This makes autoregressive models ideal for text generation, chatbots, story writing, dialogue systems etc.

Comparison and Applications While masked and autoregressive models share the same architectural foundation, their objectives lead to distinct behaviors. MLM-based models (e.g., BERT, RoBERTa) produce context-rich bidirectional representations and are well-suited for understanding and discriminative tasks. Autoregressive models (e.g., GPT-2, GPT-3, LLaMA) instead specialize in generative tasks, excelling in language modeling, completion, and reasoning through text continuation.

In this work, we focus primarily on autoregressive models trained with the next-token prediction objective, as they form the basis of the large language models studied in our experiments.

Indeed, as we will discuss in the next chapter, autoregressive language models suffer from a fundamental generalization limitation known as the **Reversal Curse**, while bidirectional models such as BERT have been shown to generalize better in similar settings.

2.2 Reversal Curse

Recent studies Berglund et al. [2023] have revealed a fundamental limitation of autoregressive language models, commonly referred to as the **Reversal Curse**. This phenomenon occurs when a model is trained on factual associations that are inherently bidirectional, but is only exposed to one direction during training. For instance, a model may learn from examples such as *"Elon Musk is the CEO of Tesla"*, yet fail to correctly infer the reversed statement *"Tesla's CEO is Elon Musk"* if it was never explicitly observed during training.

Empirically, models trained under such conditions exhibit a complete failure to generalize to the unseen direction of the fact. This behavior has been consistently observed across different architectures and datasets, and appears to be largely independent of model size or training corpus scale.

The implications of this finding are both practical and conceptual. From an applied perspective, addressing the Reversal Curse by explicitly adding reversed statements to the training corpus would require identifying all missing fact directions and potentially doubling dataset size, an evidently inefficient and impractical solution for large-scale models. From a theoretical standpoint, the phenomenon exposes a significant generalization gap in current autoregressive architectures,

calling into question their ability to represent relational symmetry and true conceptual understanding.

Interestingly, this limitation contrasts sharply with human learning: humans naturally acquire the ability to infer reversed relational statements (e.g., understanding that if A is the parent of B, then B is the child of A) at a very early age, typically by two or three years old. The fact that large language models fail at such a seemingly trivial form of reasoning highlights a fundamental divergence between statistical learning and human-like generalization, and makes the Reversal Curse an especially compelling subject of study.

In the next section, we summarize the main studies that have investigated the causes of the **Reversal Curse**. We organize the existing literature according to the key hypotheses proposed to explain the phenomenon and the methodological approaches adopted to mitigate it. In particular, we discuss works that focus on modifying the training objective, altering the model architecture, or introducing auxiliary mechanisms to encourage bidirectional reasoning.

It is worth noting that many of these approaches aim primarily at *solving* the Reversal Curse empirically, by improving generalization performance on reversed examples, rather than providing a principled *explanation* of why the phenomenon arises in the first place. While such interventions may achieve partial success, they often come at the cost of deviating from standard autoregressive modeling practices, which are known to perform exceptionally well across a wide range of natural language processing tasks.

The main limitation of these empirical fixes lies in their **impracticality**: adopting modified architectures or alternative loss functions at scale is rarely feasible, especially for large language models pre-trained with conventional autoregressive objectives. A comprehensive theoretical understanding of the Reversal Curse, which is currently lacking, would therefore be invaluable, not only for designing more targeted remedies, but also for uncovering fundamental constraints and inductive biases inherent to Transformer-based architectures. Such an explanation could further yield insights that extend beyond the Reversal Curse itself, illuminating broader principles about how these models represent and generalize relational information.

2.2.1 The NTP Failure Hypothesis

Since the **Reversal Curse** was identified as a major limitation in the generalization capabilities of large language models (LLMs), numerous studies have sought to explain its underlying causes and to propose strategies to mitigate its effects. One of the most prominent explanations in the literature attributes the phenomenon to the intrinsic limitations of the **Next Token Prediction (NTP)** objective used in autoregressive models.

The NTP objective trains a model to predict the next token in a sequence, given all preceding tokens. Formally, for a token sequence $x = (x_1, x_2, \ldots, x_T)$, the model maximizes the following likelihood:

$$P_\theta(x) = \prod_{i=1}^{T} P_\theta(x_i \mid x_{<i}), \tag{2.29}$$

where $x_{<i} = (x_1, \ldots, x_{i-1})$ denotes the prefix context. This formulation enforces a strictly *causal* dependency structure: at each step, the model only conditions on past tokens and never on future ones. While this directional objective has proven highly effective for generative modeling, enabling fluent text continuation and coherent long-range generation, it implicitly biases the model toward the unidirectional flow of information present in the training data.

In the context of the Reversal Curse, this directional bias becomes a central issue. When trained on statements such as *"Elon Musk is the CEO of Tesla"*, the model optimizes for high likelihood of the continuation "CEO of Tesla" following "Elon Musk." However, the reversed association *"Tesla's CEO is Elon Musk"* requires reasoning in the opposite direction, i.e., estimating $P(A \mid B)$ rather than $P(B \mid A)$, which the autoregressive objective does not explicitly encourage or expose during training.

Moreover, the asymmetry of natural language distributions, governed by **Zipf's law**, amplifies this imbalance. Forward statements, typically following the more frequent "concept-to-definition" pattern, occur vastly more often than their reversed counterparts, which are rarer and more diverse. Consequently, even if both relational directions exist in the dataset, the model tends to overfit to the dominant forward direction and fails to generalize bidirectionally.

This perspective, commonly referred to as the **NTP Failure Hypothesis**, suggests that the Reversal Curse is not merely an artifact of insufficient data or capacity, but a structural limitation of the autoregressive training paradigm itself. The hypothesis thus motivates exploring alternative or complementary objectives that can symmetrize learning, such as bidirectional training, contrastive losses, or explicit relation inversion tasks.

2.2.2 Data Augmentation Strategies

Supporting the NTP objective failure hypothesis, Golovneva et al. [2024] proposed a training scheme called **reverse training** to mitigate the Reversal Curse. The method augments the pre-training corpus by including each text sample both in its original left-to-right order and in a reversed (right-to-left) order. During reversal, the authors preserve the internal word order of multi-word entities (e.g., "Mary Lee Pfeiffer" remains unchanged rather than being reversed to "Pfeiffer Lee Mary"), a crucial design choice that prevents semantic corruption of named entities and allows the model to distinguish meaningful units of information. Without entity preservation, reversing entity tokens would destroy syntactic coherence and confound relational directionality with internal token reordering.

Method. Formally, for each training sample x_i, a reversed version $\overleftarrow{x_i} = \text{REVERSE}(x_i)$ is generated, where

$$\mathcal{D}^{\text{rev}}_{\text{train}} = \{x_i\}_{i=1}^{N} \cup \{\overleftarrow{x_i}\}_{i=1}^{N},$$

doubling the effective training corpus. Variants of the reversal function include token reversal, word reversal, entity-preserving reversal, and random segment reversal. The model is then trained autoregressively on this combined dataset, effectively learning both "forward" and "reversed" language modeling tasks.

Findings. Reverse training substantially mitigates the Reversal Curse across both synthetic and real-world evaluations without impairing forward performance. Table 2.1 summarizes their main results.

Building on similar motivations, Guo et al. [2024] proposed Semantic-aware Permutation Training (SPT), a method designed to alleviate the reversal curse by reordering training data while preserving structural semantics. Their central idea is that the reversal failure of causal language models primarily arises from their limited ability to predict antecedent tokens, that is, words occurring before the current position, during training. To address this, they introduce controlled permutations of training sentences that expose the model to reversed and mixed word orders without disrupting semantic coherence.

Specifically, they first employ an auxiliary language model to segment each training sentence into semantically meaningful chunks (e.g., entities or short phrases). These chunks are then

Table 2.1: Key results from Golovneva et al. [2024]. Reverse training substantially improves reversal performance while preserving forward accuracy.

Experiment	Training	Forward (%)	Reverse (%)
Symbolic reverse	Standard	100	0
	Entity-preserving	100	100
	Random segment	100	100
Biography (bioS/bioR)	Standard	–	0
	Entity-preserving	≈99	99
Real-world (Celebrity)	Standard	1.6	0.9
	Reverse training	0.8	3.6–10.4
Fictitious facts (7B)	Standard	80.3	0.0
	Reverse fine-tuning	78.3	99.0

reordered with different probabilities into the original, reversed, or randomly permuted order, ensuring that multi-word entities remain intact (e.g., "Mary Lee Pfeiffer" is not split into "Pfeiffer Lee Mary"). Preserving entities in this process is crucial: random token-level shuffling, as earlier permutation studies showed, can destroy phrase-level meaning and hinder the model's capacity to learn coherent representations.

The authors hypothesize, and empirically confirm, that the reversal curse stems less from a failure of relational understanding (e.g., grasping that "parent" and "child" are symmetric roles) and more from the directional bias introduced by left-to-right next-token prediction. Their experiments, which disentangle the effects of word order and relational phrasing (e.g., comparing "A is B's child" vs. "A's parent is B"), show that modifying the relation type has only minor effects, while reversing the entity order drastically affects performance. Thus, SPT helps the model develop a bidirectional understanding of associations while maintaining high forward accuracy, effectively addressing what we define as the Next Token Prediction (NTP) Failure Hypothesis.

2.2.3 Alternative Optimization Objectives

Beyond methods that rely on token rearrangement or data augmentation, Lv et al. [2024] propose that the training objective itself plays a pivotal role in the emergence of the Reversal Curse. Specifically, they observe that models such as GLM [Du et al., 2021], which are trained using the *autoregressive blank infilling* (ABI) objective rather than the standard next-token prediction (NTP) objective, exhibit a notable resistance to the phenomenon. Unlike NTP, where each token can only attend to its preceding context, ABI allows masked tokens to attend to both their left and right contexts, implicitly training the model to estimate both $P(b \mid a)$ and $P(a \mid b)$. This bidirectional conditioning naturally mitigates the directional bias responsible for the Reversal Curse.

To validate this hypothesis, the authors fine-tune both GLM and Llama models on a synthetic name-to-description dataset introduced by Berglund et al. [2023]. They find that while Llama and GLM trained with NTP achieve nearly 0% accuracy when tested on reversed examples, GLM trained with ABI maintains over 70−80% accuracy in both forward and reverse directions. Motivated by this observation, they further adapt causal models such as Llama to support ABI-like training through a novel objective termed **BIdirectional Causal language model Optimization (BICO)**. BICO modifies the causal attention mechanism so that tokens can attend to both preceding and succeeding positions during training, while reverting to standard causal attention during inference to preserve generative ability. The method combines ABI with NTP under a

controlled ratio, effectively balancing bidirectional understanding with autoregressive fluency.

Experimental results demonstrate that BICO-tuned Llama models largely overcome the Reversal Curse, achieving up to **70% improvement** on reversed reasoning tasks, without degrading their performance on standard forward tasks. Moreover, the benefits extend to downstream applications such as mathematical reasoning (e.g., GSM8k) and translation, suggesting that bidirectional fine-tuning not only alleviates reversal failures but also enhances general reasoning and compositionality in causal LLMs.

Formally, the BICO objective combines the standard Next Token Prediction (NTP) loss with an *Autoregressive Blank Infilling* (ABI) loss that enables bidirectional conditioning during training. Given an input sequence $x = (x_1, x_2, \ldots, x_T)$, a subset of tokens $M \subseteq \{1, \ldots, T\}$ is randomly selected as blanks. The ABI loss encourages the model to predict each masked token $x_i \in M$ from both its left and right context:

$$\mathcal{L}_{\text{ABI}} = -\sum_{i \in M} \log P_\theta(x_i \mid x_{<i}, x_{>i}), \tag{2.30}$$

where $x_{<i}$ and $x_{>i}$ denote the tokens preceding and following position i, respectively. This objective differs from classical masked language modeling (MLM) in that the model still operates in an autoregressive fashion: masked spans are predicted sequentially rather than in parallel, preserving training stability and compatibility with generative inference.

To balance bidirectional reasoning with causal fluency, the final BICO loss linearly combines ABI with the standard autoregressive NTP loss:

$$\mathcal{L}_{\text{BICO}} = \lambda\, \mathcal{L}_{\text{ABI}} + (1 - \lambda)\, \mathcal{L}_{\text{NTP}}, \tag{2.31}$$

where $\lambda \in [0, 1]$ controls the strength of bidirectional training. During inference, the model reverts to standard left-to-right decoding, ensuring compatibility with autoregressive generation.

This hybrid formulation allows causal models such as Llama to internalize relational information in both directions without sacrificing their forward generation capabilities. As a result, BICO-trained models demonstrate strong bidirectional generalization and significantly reduced sensitivity to sequence directionality, further supporting the hypothesis that the Reversal Curse originates from the unidirectional nature of the NTP objective rather than a fundamental limitation of the model architecture.

The Reversal Curse has also been examined in the context of **model editing**, a field concerned with efficiently updating or correcting factual knowledge stored within large pretrained language models (LLMs). Model editing aims to modify a model's internal representations so that it reflects new or corrected information, without requiring full retraining or extensive fine-tuning.

The growing use of LLMs in practical situations means their information will, eventually, become irrelevant or incorrect. For instance, when political leadership changes, the models might still respond the president of the United States is the previous one in office. Retraining the whole model from scratch every time one fact is changed is not only inefficient and computationally expensive but also environmentally damaging, particularly when the model has already developed a considerable linguistic and conceptual foundation. Hence, there is a strong motivation for developing techniques that can perform *localized edits*, modifying only the relevant internal representations while preserving the rest of the model's behavior.

To give a concrete example, we turn to one of the most impactful and conceptually straightforward model-editing methods: Rank-One Model Editing (ROME) Meng et al. [2022]. Being a causal model editing, ROME attempts to find and alter the specific internal model parameters

tied to a particular memory. ROME's key insight is that, for example, the model might store the fact "Paris is the capital of France" in a particular sub-area of the model's feedforward layers, and more specifically, in the MLP modules of the transformer blocks. By determining the linear transformation that connects a subject representation ("Paris") to an object ("France"), ROME performs a low-rank (usually rank one) update to the relevant weight matrix and overwrites the old association to a new one. This precision enables the model to add or correct individual facts without extensive fine-tuning or retraining. ROME, therefore, serves as a fundamental editing technique for large language models.

Empirically, ROME has been shown to perform precise factual edits while preserving global performance across unrelated tasks, making it a foundational technique for efficient and interpretable model editing in large language models.
We provide below a brief mathematical summary of the technique.
Consider a Transformer with an MLP block at layer ℓ of the form

$$h_{\text{out}}^{(\ell)} \;=\; h_{\text{in}}^{(\ell)} \;+\; W_{\text{out}}^{(\ell)}\,\sigma\big(W_{\text{in}}^{(\ell)}\,h_{\text{in}}^{(\ell)} + b_{\text{in}}^{(\ell)}\big),$$

where σ is a pointwise nonlinearity. Let the *edit prompt* place the subject token (e.g., "Paris") at a chosen position so that, during a forward pass, the MLP's *key* (post-activation) vector at layer ℓ is

$$k \;=\; \sigma\big(W_{\text{in}}^{(\ell)}\,h_{\text{in}}^{(\ell)} + b_{\text{in}}^{(\ell)}\big) \;\in \mathbb{R}^{d_{\text{mlp}}}.$$

ROME seeks to overwrite the mapping $k \mapsto W_{\text{out}}^{(\ell)}k$ so that it points toward a *target residual direction* $v^\star \in \mathbb{R}^{d_{\text{model}}}$ encoding the new object (e.g., the residual direction aligned with the token "France"). A standard choice is

$$v^\star \;=\; \alpha\,U^\top e_{\text{obj}},$$

where U is the unembedding matrix, e_{obj} is the one-hot vector of the target token, and $\alpha > 0$ is a scale chosen by a small calibration (so the edited token is confidently predicted).

ROME computes a minimal-change update ΔW to the MLP output matrix $W_{\text{out}}^{(\ell)}$ by solving the constrained least-squares problem

$$\begin{aligned} \min_{\Delta W} \quad & \|\Delta W\|_F^2 \\ \text{s.t.} \quad & (W_{\text{out}}^{(\ell)} + \Delta W)\,k \;=\; v^\star. \end{aligned} \tag{2.32}$$

Let $r \;=\; v^\star - W_{\text{out}}^{(\ell)}k$ be the required change at k. The minimum-Frobenius-norm solution is a *rank-one* update:

$$\Delta W \;=\; \frac{r\,k^\top}{k^\top k}.$$

Finally, the edited weight is

$$W_{\text{out}}^{(\ell)} \;\leftarrow\; W_{\text{out}}^{(\ell)} + \Delta W, \qquad \text{so that} \qquad \big(W_{\text{out}}^{(\ell)} + \Delta W\big)k \;=\; v^\star.$$

Practical refinements.

- *Locality preservation.* To keep unrelated behavior intact, ROME optionally adds a small "neighborhood" set of prompts $\{x_j\}$ and prevents output for drifting excessively.
- *Edit site selection.* The edited layer ℓ is chosen where causal tracing/gradient attribution (we will explore this technique in our later section on interpretability methods) indicates the subject $\rightarrow$ object mapping is realized (typically an early/mid MLP).

Coming back to the Reversal Curse, recent studies have shown that even after a model is successfully edited to store a new fact (for example, inserting "A is B"), it often fails to retrieve the corresponding reversed statement ("B is A"). This suggests that the Reversal Curse is not just a training-time limitation, but also persists inside the model's edited representations.

This finding has led to the design of new editing objectives and strategies aimed at enforcing relational symmetry, one of the most notable being the BIRD framework, which we discuss next.

2.2.4 Shortcut Learning

A different perspective on the NTP Failure Hypothesis is offered by Zhang et al. [2024], who argue that autoregressive LLMs may primarily rely on surface-level word co-occurrences rather than genuinely capturing factual associations.
Deep Neural Networks (DNNs) have been shown to rely on shortcuts rather than learning the true underlying distributions, often a more complex task, in various domains such as computer vision, natural language processing, and reinforcement learning [Geirhos et al., 2020]. This behavior, therefore, is not unprecedented. The authors also demonstrate that when models are encouraged to learn indirect factual associations, they exhibit improved generalization capabilities.
This is reflected in significantly higher performance across various downstream tasks, when the model is prompted with questions phrased in the reverse direction relative to the training data. An illustration of an indirect factual association is presented below:

The capital city of France is Paris.	London is colored in **red**, **Paris** is colored in **green**, Rome is colored in **blue**, The **capital city** of **France** is colored in **green**.

Although the authors demonstrate that direct associations encourage the model to rely on shortcuts by capturing co-occurrence statistics rather than true associations, such associations can still lead to weak learning of factual information.
The primary concern is that, when optimizing for statistical patterns rather than genuine relations, the loss quickly converges and gradients diminish, impeding further learning. To address this, the authors propose an intriguing approach involving the resetting of model parameters.
Empirical evidence also suggests that these methods help mitigate another generalization failure in LLMs, known as the two-hop curse [Balesni et al., 2024].

2.2.5 Input-Output Mismatch Hypothesis

The Two-Hop Curse refers to the inability of large language models (LLMs) to integrate separately learned facts in the absence of explicit reasoning strategies, such as chain-of-thought (CoT) prompting or similar methods.
For instance, a model may correctly learn the fact A $\rightarrow$ B ("John Lennon performed Imagine") and B $\rightarrow$ C ("John Lennon's spouse is Yoko Ono"), yet fail to infer the composite relation A $\rightarrow$ C ("The spouse of the performer of Imagine is Yoko Ono") unless guided by an explicit reasoning chain.
The Two-Hop Curse appears to be a generalization failure closely related to the Reversal Curse, as both phenomena are strongly correlated: methods that mitigate one often show improvements on the other as well.
In particular, the findings suggest that both limitations may arise from a shared underlying deficiency: the inability of LLMs to perform reasoning within their latent representation space.

Through a series of experiments, the authors demonstrate that LLMs are capable of multi-hop reasoning without Chain-of-Thought (CoT) when the relevant facts are co-located either within the same document or, even more effectively, within the same prompt.
To encourage such compositional reasoning, they attempt to enforce locality by training the earlier layers of the model on fact A and the later layers on fact B, thereby investigating whether layer-wise sequentiality enhances two-hop performance.
Additionally, they introduce an auxiliary loss designed to compel the model to form a bridge representation in latent space, facilitating the composition of separate facts.
Neither of the two approaches leads to improved performance on two-hop question answering, suggesting that the two-hop curse may be difficult to overcome without a deeper understanding of its underlying causes.
To this end, the authors propose the **Input-Output Mismatch Hypothesis**, which suggests that the two-hop curse, along with other generalization failures of LLMs, may stem from a fundamental discrepancy between the input and output representations of a concept.
Wang and Sun [2025] affirm this hypothesis and, in exploring the root causes of the reversal curse, identify one contributing factor as the model's failure to bind an entity when it switches roles between a perceived subject and a predicted object.
They conduct an experiment demonstrating that transformers are indeed capable of learning reverse relations at the abstract concept level.
Specifically, by constructing a setup in which each concept (entity or relation) is represented by its own learnable embedding, they show that the model achieves high performance on reversal tasks.
This leads the authors to hypothesize that the core issue lies in the inconsistency of concept representations across different contexts.
As a result, they identify two main reasons why transformers fail to capture reverse relations in more realistic settings:

1. Inconsistency: The authors argue that LLMs lack the inductive bias needed to form consistent representations of concepts.
   In the context of reversal, this results in a failure to bind entity representations reliably when their roles shift between subject and object.

2. Entanglement of Concept Representations: The authors contend that transformers struggle to disentangle abstract concepts from surface-level patterns.
   As a result, modifying the representation of one entity or relation during training can inadvertently alter the representation of unrelated concepts, thereby hindering generalization.

   To illustrate this, suppose we have two latent concepts a and b, with MLP activations α and β, respectively. Let the output projection matrix be V where v_i is its i-th column, and hence we can express the two concepts as:

   $$a = \sum_i \alpha_i v_i, \qquad b = \sum_i \beta_i v_i$$

   The desired update directions for these two concepts are given by:

   $$-\frac{\partial \mathcal{L}}{\partial a} \quad \text{and} \quad -\frac{\partial \mathcal{L}}{\partial b}$$

   Assuming that α and β remain constant, the gradient of the loss with respect to each basis vector v_i is:

   $$\frac{\partial \mathcal{L}}{\partial v_i} = \alpha_i \frac{\partial \mathcal{L}}{\partial a} + \beta_i \frac{\partial \mathcal{L}}{\partial b}$$

After one step of gradient descent with learning rate η, the concept representations are updated as:

$$\Delta a = \sum_i \alpha_i \left(v_i - \eta \frac{\partial \mathcal{L}}{\partial v_i} \right) - a = -\eta \|\alpha\|^2 \frac{\partial \mathcal{L}}{\partial a} - \eta \alpha^\top \beta \frac{\partial \mathcal{L}}{\partial b}$$

$$\Delta b = \sum_i \beta_i \left(v_i - \eta \frac{\partial \mathcal{L}}{\partial v_i} \right) - b = -\eta \|\beta\|^2 \frac{\partial \mathcal{L}}{\partial b} - \eta \alpha^\top \beta \frac{\partial \mathcal{L}}{\partial a}$$

As we can see, the update of each concept is not entirely aligned with its gradient but rather includes an entanglement factor proportional to the dot product $\alpha^T \beta$. For this reason, unrelated entities that share one surface-level token might be affected by this problem (for example Emma Stone and Mary Stone might be completely unrelated despite sharing the same last name, yet be influenced from each other). To address these issues, the authors adopt an approach inspired by the JEPA architecture [LeCun, 2022] to promote consistency within the embedding space.
Additionally, they introduce memory layers [Sukhbaatar et al., 2015]: ultra-wide hidden layers with top-k sparsity and softmax activations, as a mechanism to mitigate the entanglement of concept representations.
Their results demonstrate that, with these modifications, transformers are able to successfully learn reverse relations and, moreover, exhibit improved generalization.
The authors validate this by conducting an experiment in which the model is tested on arithmetic and simple mathematical reasoning tasks, with empirical evidence showing that the modified models significantly outperform standard (vanilla) transformers.

2.3 Interpretability techniques

Although deep learning architectures are often difficult to interpret and may develop internal algorithms that are not easily explainable from a human perspective, as have already discussed, studying their behaviour can be extremely beneficial to understand their potential and limitations.
Analyzing the deep causes of why limitations like the reversal curse occur can make more clear what we can and can not expect from these models, how to improve their training and potentially even design methods that perform better on the tasks of interest.
Therefore, in this work, we employ such interpretability techniques to investigate why the reversal curse affects autoregressive models in realistic settings and how the model can overcome it in the synthetic ones, hoping to bridge the gap between the two.
We now briefly introduce some of the main methods used in this field.

2.3.1 Causal Tracing

The causal tracing methodology introduced by Meng et al. [2022] allows precise localization of factual associations inside autoregressive LLMS GPT-like architectures.

This technique is primarily used to identify which layers and components store a given fact, after which the authors apply the ROME (Rank-One Model Editing) method to overwrite these representations with new factual information, as previously discussed.

The core idea is that the authors treat the transformer computational graph as a causal graph Pearl [2022].
Although other methods, like integrated gradients and saliency maps, can be used to identify correlation between outputs and activations, but do not test *causation*: a high gradient or activation value does not imply that the variable causes the output.

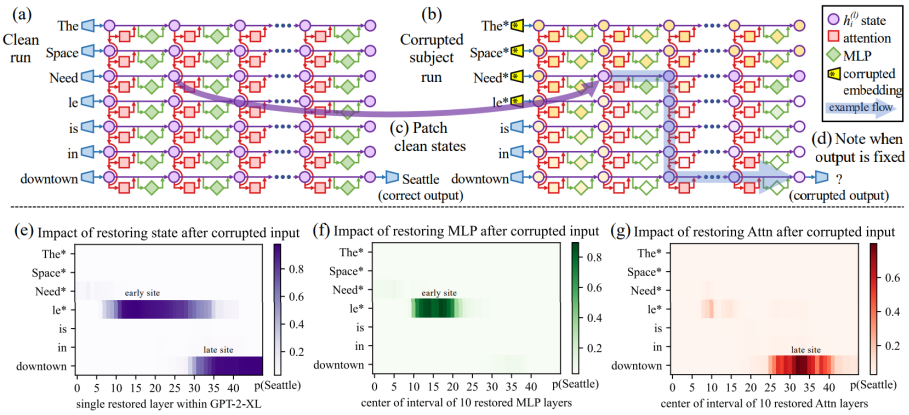


Figure 2.1: Causal Tracing

On the other hand, causal tracing focuses on measuring the *indirect causal effect* of internal activations on model's predictions.

Experimental Setup A simple factual association can be represented as a triple $t = (s, r, o)$, where s, r, and o denote the *subject*, *relation*, and *object*, respectively. Given a natural-language prompt $p(s, r)$ expressing the subject and relation, the model is expected to generate the correct object o as the next token.

To measure the *causal contribution* of each hidden state $h_i^{(l)}$, Meng *et al.* **?** perform three controlled forward passes through the transformer model G:

(1) Clean run. The prompt $p(s, r)$ is given to the model and all hidden activations $\{h_i^{(l)} \mid i \in [1, T],\ l \in [1, L]\}$, which are responsible for generating the current object token o, are saved.

(2) Corrupted run. The subject token representation is intentionally corrupted, usually applying gaussian noise to its embedding:

$$h_i^{(0)} \leftarrow h_i^{(0)} + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \nu I). \tag{2.33}$$

This corruption removes information about the subject s, causing the model to fail in predicting the correct object o.

(3) Corrupted-with-restoration run. The network is executed as in the corrupted case, but at a selected layer–position pair (l, i), the corresponding corrupted activation is replaced (with a forward hook) by its clean counterpart from run (1):

$$h_i^{(l)} \leftarrow h_{i,\text{clean}}^{(l)}. \tag{2.34}$$

After this intervention, the model continues the forward pass normally, allowing the restored activation to propagate downstream.

Quantifying Causal Effect Let:

- $P[o]$: probability of emitting the correct object o in the *clean* run.
- $P^*[o]$: probability in the *corrupted* run.

- $P^{*,\,\text{clean } h_i^{(l)}}[o]$: probability in the *corrupted-with-restoration* run, where $h_i^{(l)}$ is restored to its clean value.

Then, the following quantities are defined:

Total Effect (TE). The degradation in performance due to corruption:

$$\text{TE} = P[o] - P^*[o]. \tag{2.35}$$

Indirect Effect (IE). The portion of the lost performance recovered by restoring a specific hidden state $h_i^{(l)}$:

$$\text{IE}(h_i^{(l)}) = P^{*,\,\text{clean } h_i^{(l)}}[o] - P^*[o]. \tag{2.36}$$

A high indirect effect indicates that the activation $h_i^{(l)}$ *mediates* crucial information required to recall the correct fact. Averaging $\text{IE}(h_i^{(l)})$ across multiple factual statements yields the *Average Indirect Effect (AIE)* per layer and token position.

2.3.2 Layerwise Decoding (Logit/Tuned Lens) to Measure Emergent Accuracy

Let a transformer G with L blocks process a length-T prompt $x_{1:T}$. Denote by $h_T^{(l)} \in \mathbb{R}^d$ the residual-stream activation at the final token after block $l \in \{0, \ldots, L\}$ (with $l = 0$ the embedding+positional input). Let $U \in \mathbb{R}^{d \times |V|}$ and $b \in \mathbb{R}^{|V|}$ be the model's unembedding and bias.[1]

Logit lens (shared decoder). Define layer-l logits by

$$z^{(l)} \triangleq h_T^{(l)} U + b \in \mathbb{R}^{|V|}, \qquad p^{(l)} \triangleq \text{softmax}\Big(z^{(l)}\Big). \tag{2.37}$$

Given the gold next token $o \in V$, the layerwise negative log-likelihood (NLL) and top-1 accuracy are

$$\mathcal{L}^{(l)} \triangleq -\log p^{(l)}[o], \qquad \text{Acc}^{(l)} \triangleq \mathbb{1}\Big[\arg\max_{v \in V} p^{(l)}[v] = o\Big]. \tag{2.38}$$

For a dataset $\mathcal{D}$ of factual prompts, we report $\overline{\text{Acc}}^{(l)} \triangleq \frac{1}{|\mathcal{D}|} \sum_{(x,o) \in \mathcal{D}} \text{Acc}^{(l)}$ (and analogously $\overline{\mathcal{L}}^{(l)}$) as a function of l (and optionally of training step). An increasing $\overline{\text{Acc}}^{(l)}$ across layers indicates progressive refinement of predictions.

Tuned lens (learned affine probes). To mitigate distribution shift between intermediate representations and the final decoding space, fit an affine probe $(A^{(l)}, a^{(l)})$ per layer on a held-out split:

$$\tilde{h}_T^{(l)} \triangleq A^{(l)} h_T^{(l)} + a^{(l)}, \qquad \tilde{z}^{(l)} \triangleq \tilde{h}_T^{(l)} U + b, \qquad \tilde{p}^{(l)} \triangleq \text{softmax}\Big(\tilde{z}^{(l)}\Big), \tag{2.39}$$

and evaluate $\tilde{\text{Acc}}^{(l)}$ as above. The tuned lens better reflects the model's "latent beliefs" layer-by-layer and is empirically more robust than the raw logit lens.

[1] If the model applies a final LayerNorm or MLP head before U, compose it into the mapping below; equivalently, work in the pre-softmax (logit) space.

When does the model "know" the answer? Define the *emergence depth* for instance (x, o) as

$$l^\star(x, o) \triangleq \min\{\, l \,:\, \arg\max_v p^{(l)}[v] = o \;\; (\text{or } \tilde{p}^{(l)})\,\}. \tag{2.40}$$

We summarize with the distribution of $l^\star$ over $\mathcal{D}$, or its mean $\mathbb{E}[l^\star]$, optionally tracked across training checkpoints to study when correct predictions first become linearly decodable.

2.3.3 Zero-Ablation of Submodules to Test Necessity

Let $\mathcal{C}$ be a collection of disjoint components of G (e.g., individual attention heads, MLP blocks, full layers). For a component $c \in \mathcal{C}$, write $f_c(h)$ for its output on input h at its insertion site (residual stream). *Zero-ablation* replaces $f_c(h)$ by the zero vector:

$$\mathrm{Ablate}_0(f_c)(h) \triangleq \mathbf{0}. \tag{2.41}$$

Let $P_{\text{clean}}[o \mid x]$ be the model's next-token distribution without ablation and $P_{\neg c}[o \mid x]$ the distribution when c is zero-ablated (all else unchanged). We quantify the necessity/importance of c for (x, o) by any of:

$$\Delta \log p_c(x, o) \triangleq \log P_{\text{clean}}[o \mid x] - \log P_{\neg c}[o \mid x], \tag{2.42}$$

$$\Delta \mathrm{CE}_c(x, o) \triangleq -\log P_{\neg c}[o \mid x] + \log P_{\text{clean}}[o \mid x] \;=\; \Delta \log p_c(x, o), \tag{2.43}$$

$$\Delta \mathrm{Acc}_c(x, o) \triangleq \mathbb{1}\Big[\arg\max_v P_{\text{clean}}[v \mid x] = o\Big] - \mathbb{1}\Big[\arg\max_v P_{\neg c}[v \mid x] = o\Big]. \tag{2.44}$$

Dataset-level importance is then $\mathbb{E}_{(x,o)\sim\mathcal{D}}[\Delta \log p_c]$ (or the fraction of accuracy drops). Ranking components by this score localizes submodules that are *necessary* for correct prediction.

Practical notes. (i) To reduce distribution shift, one may use *mean ablation*: replace $f_c(h)$ by its dataset mean $\bar{f}_c \triangleq \mathbb{E}[f_c(h)]$ instead of $\mathbf{0}$. (ii) For attention heads, ablation can be applied to the head output before residual addition; for MLPs, set their output to zero; for full layers, skip both attention and MLP residual adds. (iii) Report per-relation or per-task aggregates to avoid masking effects from easy instances.

Chapter 3

Results

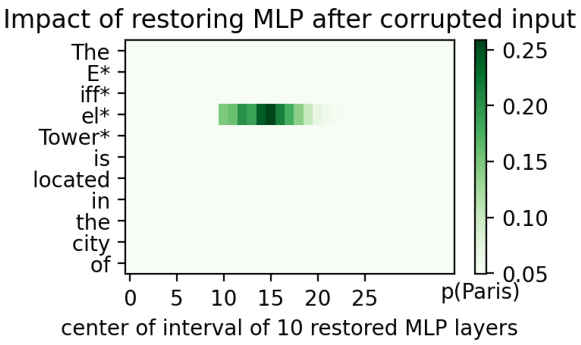
3.1 ROME and the Reversal Curse

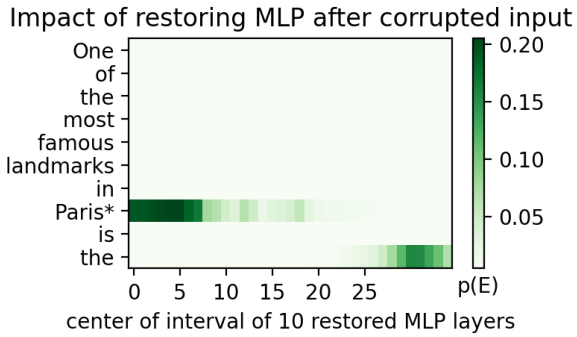
One of the fundamental questions when studying the reversal curse is whether the model stores knowledge about a fact in a single location,shared across both directions, or whether it maintains two distinct sets of parameters, one for each direction of the same fact.
Ideally, we would want the model to store each piece of factual knowledge in one place, accessible regardless of direction or phrasing.
This aligns with how humans recall factual information: we represent a single concept and any linguistic variation expressing the same meaning activates the same underlying memory or representation.

Intuitively, this does not seem to hold for models affected by the reversal curse.
By definition, if the model can recall a fact only when prompted in one specific direction, it is reasonable to assume that the two directions rely on different internal parameters.
However, previous studies have shown that LLMs may, in some cases, *possess* the correct information even in the "failed" direction, but are unable to produce it during generation.

This occurs because autoregressive LLMs are designed to generate text that aligns with the distribution of human language, they aim to output the most probable token given the context, not necessarily what they "know" to be true.
Therefore, an alternative hypothesis is that the model might in fact rely on the same underlying parameters for both directions, but fails to express one of them due to statistical biases in token likelihood.
We will see in later sections that there are cases where the model assigns a low probability to the correct token when prompted with the reversed sentence, yet still selects the right answer from a subset in multiple-choice settings. This highlights that the underlying knowledge is present somewhere in the model, and that the problem might lie primarily in retrieval rather than storage.

To investigate this, we applied the causal tracing technique introduced earlier in the interpretability section.
Using a pretrained GPT-2 model, already containing substantial general knowledge, we traced where the model stores specific factual associations.
For each fact, we prompted the model in both directions and compared the internal activations to determine whether the representations overlap. We found that, on average, the model stored the two directions of each fact in distinct submodules, relying on separate sets of weights.
In most cases, the corresponding representations were located in entirely different layers of the model, providing strong evidence in support of the first hypothesis.

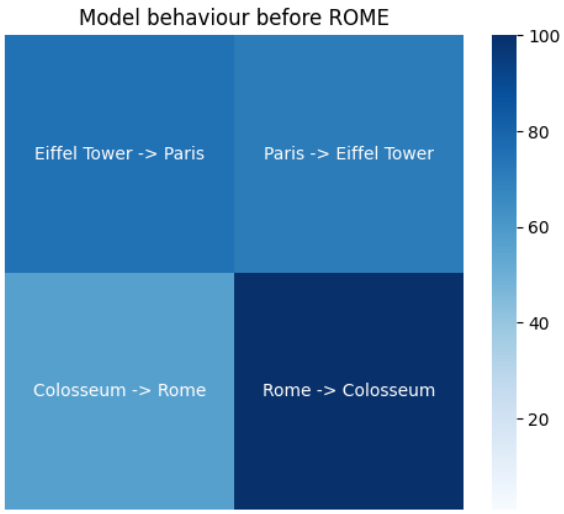


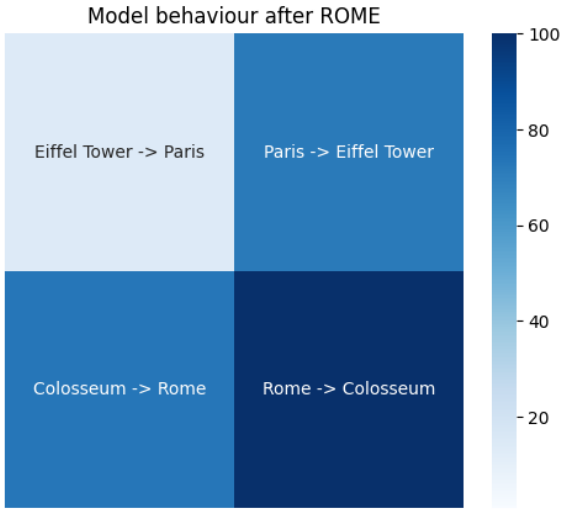


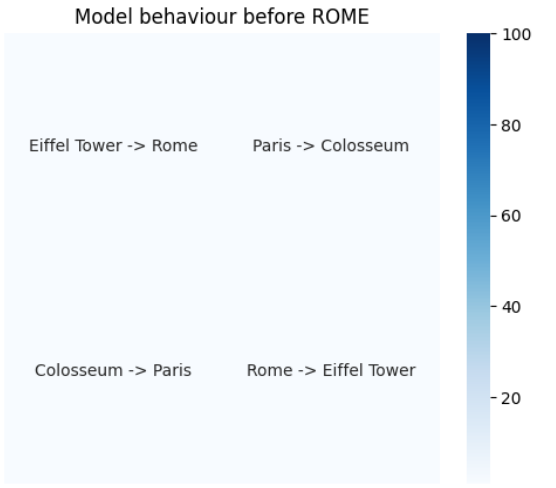
We further validated this finding by editing the model using the Rank-One Model Editing (ROME) technique.
If the model truly used the same set of weights for both directions of each fact, then a single ROME edit, which applies a rank-one update to the model parameters, should automatically update the opposite direction "for free."
For instance, suppose we modify the sentence *"The Eiffel Tower is the most important landmark in Paris"* to *"The Eiffel Tower is the most important landmark in Rome"* using ROME.
If the underlying parameters represent the same concept, we should automatically obtain the reversed statement *"Rome's most important landmark is the Eiffel Tower"*.
At the same time, previously stored knowledge that contradicts the new edit, such as *"The Eiffel Tower is the most important landmark in Paris"* or *"Paris' most important landmark is the Eiffel Tower"*, should decrease in score.

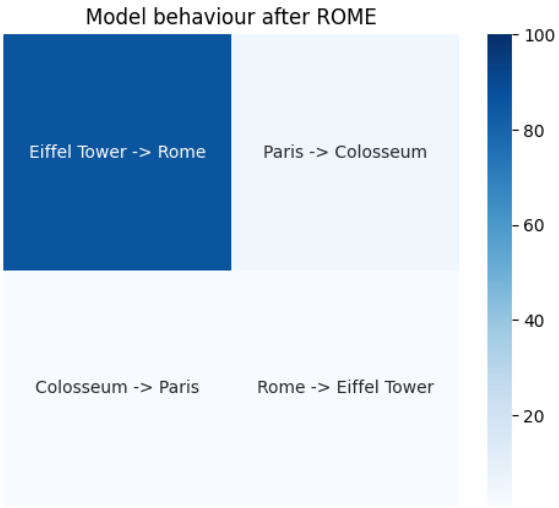
To test this, we augmented each prompt to a meaningful sample size (100 instances) and evaluated which directions the model was able to generate before and after the ROME intervention. In practice, we observed that ROME modifies the factual association only in one direction, leaving inconsistencies in the opposite one.

This result further supports the hypothesis that the two directions are stored in different parameters or submodules, and is consistent with our later findings on the model's difficulty in forming stable and coherent latent concept representations.


Model behaviour before ROME
Eiffel Tower -> Paris
Paris -> Eiffel Tower
Colosseum -> Rome
Rome -> Colosseum
100
80
60
40
20


Model behaviour after ROME
Eiffel Tower -> Paris
Paris -> Eiffel Tower
Colosseum -> Rome
Rome -> Colosseum
100
80
60
40
20


Model behaviour before ROME
Eiffel Tower -> Rome
Paris -> Colosseum
Colosseum -> Paris
Rome -> Eiffel Tower
100
80
60
40
20


Model behaviour after ROME
Eiffel Tower -> Rome
Paris -> Colosseum
Colosseum -> Paris
Rome -> Eiffel Tower
100
80
60
40
20

3.2 Indirect Training

3.2.1 Introduction and Initial results

In this section, we build upon the approach proposed by Zhang et al. [2024].
The authors propose that creating datasets based on indirect associations, rather than relying solely on straightforward factual statements, which is the common practice, could force large language models to genuinely learn about entities and their relationships, rather than depending on word-level statistical patterns or other superficial shortcuts.
To test their hypothesis, the authors create two custom synthetic training datasets, called "narrative" and "referencing".
The narrative dataset is a collection of simple fictional facts, which include three entity categories (country, city, animal) and two relations (iscapital and famousfor). The dataset is created from simple templates, which augment the same fact 10 times to represent natural variation of the narrative text.

The capital city of Andoria is Copperton.
Copperton is the capital of Andoria.
Andoria's capital city is Copperton.
Copperton serves as the capital of Andoria.
...
If you love dolphin, Fironzia is the place to be.
The city Sunfield is famous for its jellyfish.
Sunfield is renowned for its jellyfish.
jellyfish is the pride of Sunfield.

On the other hand, the referencing dataset includes the same facts, but never in a straightforward manner. The facts are expressed implicitly and indirectly, often linked through a "bridge" entity, such as a color, that connects the two main entities without stating their relationship explicitly.

Salton is colored in red, Fironzia is colored in blue, Copperton is colored in green, and Clearview is colored in yellow. Therefore, the capital city of Andoria is colored in green.
Salton is colored in red, Fironzia is colored in blue, Cymperia is colored in green, and Copperton is colored in yellow. Therefore, the capital city of Andoria is colored in yellow.
Pyrendi is colored in red, Uxendal is colored in blue, Brindocor is colored in green, and Copperton is colored in yellow. Therefore, the capital city of Andoria is colored in yellow.
...
What is the capital city of Andoria? A. Copperton B. Salton C. Brindocor D. Ashbourne
Answer: A
What is the capital city of Andoria? A. Uxendal B. Copperton C. Greystone D. Ashbourne
Answer: B
What is the capital city of Andoria? A. Cymperia B. Copperton C. Riventhel D. Willowcreek
Answer: B

Subsequently, two models, finetuned on the narrative and referencing datasets, are tested on various evalutation dataset, to capture their overall generalization capabilites.
The tests include quantifying the capability of the models to use the information from the training corpus to answer more complex queries at inference time (indirect reasoning and 2-hop reasoning), assessing the models' ability to retrieve answers from questions "in the reverse order" with respect to the training corpus (Reverse QA) and the performance when forced to answer among 4 possible options (multiple choice).

Table 3.1: Performance of LLaMA 3 8B on various reasoning tasks using different training data.

Training data	QA	Multiple choice	Reverse QA	Indirect reasoning	2-hop reasoning
Narrative	**100**	58.2	52.5	65.0	38.8
Referencing	**100**	**98.8**	**97.5**	**84.0**	**92.5**

The results from the paper seem promising and suggest a possible connection between the reversal curse the shortcut learning phenomenon.
However, we express our concerns regarding the data contained by the two datasets: both formats include examples where the two entities of a fact appear in both directions.
Below we show examples of this for both the narrative dataset and the referencing one.

Narrative:
The capital city of Andoria is Copperton. ($Country \cdots City$)
Copperton is the capital of Andoria. ($City \cdots Country$)

Referencing:
What is the capital city of Andoria? A. Copperton B. Orilixis C. Brindocor D. Willowcreek Answer: A ($Country \cdots City$)
Copperton is colored in red, Cymperia is colored in blue, Tivarion is colored in green, and Salton is colored in yellow. Therefore, the capital city of Andoria is colored in red. ($City \cdots Country$)

As pointed out by Golovneva et al. [2024] and Guo et al. [2024], re-arranging the tokens in the reverse direction is considerably effective in mitigating the reversal curse.
The authors do not deem this to be problematic for their objective, since they aim at assessing the general understanding and depth of the model trained on indirect associations, and the reverse recalling is just one task in their benchmark.
Moreover, their goal is to model a general scenario, where the data probably includes many phrasings of the same concept, incorporating different wordings and directions.
However, given our objective is to quantify the effect of the indirect training on nursing the reversal curse, we decide to filter the two datasets so that the entities only appear in one direction (either $Country \cdots City$ or $City \cdots Country$).

We call the resulting four datasets:
narrative_filtered, *referencing_filtered* ($Country \cdots City$)
narrative_filtered_reverse, *referencing_filtered_reverse* ($City \cdots Country$)

We notice that the performance of models fine-tuned on these datasets and tested on the reverse direction (*Eval_reverse* for *narrative_filtered* , *referencing_filtered* and *Eval_QA* for *narrative_filtered_reverse* , *referencing_filtered_reverse*) drops drastically compared to the results obtained by the original study.

Table 3.2: Performance of LLaMA 3 8B when fine-tuned on filtered dataset.

Training data	QA	Multiple choice
narrative_filtered_reverse	2.5	30
referencing_filtered_reverse	**15**	**75**

The model trained on the *narrative_filtered_reverse* dataset is unable to break the reversal curse, achieving a perfomance comparable to chance level on the QA task.
On the other hand, the model trained on the *referencing_filtered_reverse* dataset achieves 15% on this task, which although being a much smaller figure compared to the first results, is almost tenfold the score from the model trained on direct facts.
These percentages rise to 30% and 75% respectively, if we consider the multiple choice task, where we compare the logits of four possible answers produced by the models.
These results undoubtedly confirm the importance of token directionality and the auto-regressive training policy among the causes of the reversal curse.
At the same time, our findings hint at the concrete possibility that, as showed originally, a richer dataset, that nudges the model towards more complex circuits and non-superficial relationships, might help significantly alleviate the reversal problem, even if not to the scale the authors claimed initially.

3.2.2 Bridge Entity Hypothesis

In this section, we focus on investigating further whether a corpus of indirect factual associations might significantly elicit reverse retrieval mechanisms in language models, like our earlier experiments were indicating.
In particular, we proceed to propose a plausible explanation of this phenomenon, linking it to previous research like the factorization curse Kitouni et al. [2024], and testing it on the examples introduced in the previous chapter.

When modeling the probability of some text (or equivalently, a sequence of tokens) we can consider all its possible factorizations to be equivalent, including the forward (left-to-right) and the reverse (right-to-left) directions.

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_{n-i+1} \mid x_{n-i+2}, \ldots, x_n)$$

However, when training a language model using the autoregressive objective (e.g., Radford et al. [2018]), the model is explicitly optimized to reflect a left-to-right factorization of the joint probability distribution. As a result, the left-to-right and right-to-left factorizations no longer yield equivalent representations of the joint distribution, introducing an inherent asymmetry in the model's learned probabilities.
During autoregressive training with a left-to-right factorization, the model learns to predict each token based solely on its preceding context. For a relational fact expressed as the sequence (e_1, r, e_2), the training signal explicitly encourages the model to map the hidden representation derived from (e_1, r) toward predicting e_2. Thus, parameters (e.g., in MLP and attention layers) are updated so that encountering (e_1, r) in the future leads to a high likelihood of e_2.

By contrast, the model is never trained to predict e_1 given (r, e_2), and therefore receives no gradient to associate (e_2, r) with e_1. Although e_2 is encoded with full access to (e_1, r), the only training signal updating e_2's representation comes from predicting the subsequent token (often punctuation or end-of-sequence), which carries minimal semantic information about the relation

back to e_1. Consequently, there is little meaningful pressure to embed the inverse relation $(e_2, r) \to e_1$. The learned representations thus exhibit a directional bias: the model becomes proficient at inferring e_2 from (e_1, r) but not the reverse.

We can witness this in the filtered version of the narrative dataset, that we have previously studied, where having the cities preceding the countries reduces to chance level the score of reversal retrieval in testing.

narrative_filtered_reverse: Willowcreek is the capital of Oceana.
Eval_QA: What is the capital city of Oceana?

When finetuning on *narrative_filtered_reverse* we never stimulate the model to generate anything meaningful from the representation of "Oceana" and this results in the model failing to predict the correct answer ("Willowcreek").

In contrast, if we explore the examples from the *referencing_filtered_reverse* dataset:

Willowcreek is colored in **red**, Fironzia is colored in blue, Sunfield is colored in green, and Dravendel is colored in yellow. Therefore, the capital city of Oceana is colored in **red**.

Tivarion is colored in red, Salton is colored in blue, Willowcreek is colored in **green**, and Dravendel is colored in yellow. Therefore, the capital city of Oceana is colored in **green**.

The model is now induced to update the representations of both entities similarly, since both e_1 (Willowcreek) and (r, e_2) (capital of Oceana) are aligned to predict the same token (red, green, etc.)
In contrast to the narrative dataset, where "Oceana" is always one of the last tokens, in referencing we have to perform a meaningful prediction afterwards (the model has to predict the correct color using the representation of "Capital of Oceana").
For this reason, we speculate that in the finetuning phase, the major contribution is given by the loss corresponding to the "correct" color token.
We call this the "Bridge Entity Hypothesis", since we claim that when predicting the same token, the model build an internal bidirectional association between two entities (in this case "Willowcreek" and "Capital of Oceana").
To test this hypothesis, our first experiment consists of masking all the tokens but the bridge one (the color that gets repeated, in the examples we have illustrated) while finetuning.
A similar score of the model finetuned in this manner would be a strong indication that the bridge token is doing most of the uplifting in training and would validate our hypothesis.

Table 3.3: Narrative vs Referencing vs Referencing Masked

Training data	**QA**	**Multiple choice**
narrative_filtered_reverse	2.5	30
referencing_filtered_reverse	**15**	**75**
referencing_filtered_reverse (masked)	7.5	57.5

Despite the performance on the masked model decreasing on both evaluation tasks, it is remarkable that its results are better than the ones from the model trained on the narrative dataset and still above chance level.

3.3 Relational Embedding Models

A variety of neural architectures have been proposed to model structured relations between entities, each defining a scoring function $f(s, r, o)$ that assigns high scores to true triples (s, r, o) and low scores to corrupted ones. These models are generally trained via margin-based ranking losses and negative sampling, where false triples are generated by replacing either the subject or the object.

Below, we briefly describe the main models we tested in our experiments: TransE Bordes et al. [2013], RotatE Sun et al. [2019], ComplEx Trouillon et al. [2016], and SimplE Kazemi and Poole [2018]. All of them embed entities and relations into a continuous vector space and learn through the same general principle:

$$\mathcal{L} = \mathbb{E}_{(s,r,o)\in\mathcal{D}} \left[\sum_{o'\in\mathcal{N}(s,r)} \max(0, \gamma + f(s, r, o') - f(s, r, o)) \right],$$

where $\mathcal{N}(s, r)$ denotes the set of negatively sampled triples and γ is a fixed margin.

3.3.1 TransE

TransE Bordes et al. [2013] models each relation r as a translation vector that should transform the subject embedding e_s close to the object embedding e_o:

$$f_{\text{TransE}}(s, r, o) = -\|e_s + e_r - e_o\|_2.$$

The training objective encourages $e_s + e_r \approx e_o$ for true triples, and the distance to be large for corrupted ones. Despite its simplicity, TransE captures one-to-one relations effectively but struggles with symmetric or many-to-many relations due to its purely additive nature.

3.3.2 RotatE

RotatE Sun et al. [2019] extends TransE into the complex plane, representing each entity as a complex vector $e = e^{\text{re}} + ie^{\text{im}}$ and each relation as a rotation in that space:

$$f_{\text{RotatE}}(s, r, o) = -\|e_s \circ r - e_o\|_2, \quad \text{with } r = e^{i\theta_r}.$$

Here, relations are modeled as phase shifts parameterized by θ_r, making the model naturally capable of representing inversion ($r^{-1} = r^*$), composition, and symmetry, since rotations can encode both additive and multiplicative structures. After each gradient step, entity embeddings are normalized to unit modulus to preserve the rotational geometry.

3.3.3 ComplEx

ComplEx Trouillon et al. [2016] also embeds entities and relations into a complex vector space but uses a Hermitian inner product as the scoring function:

$$f_{\text{ComplEx}}(s, r, o) = \text{Re}\left(\langle e_s, r, \overline{e_o}\rangle\right) = \text{Re}\left(\sum_k e_{s,k}\, r_k\, \overline{e_{o,k}}\right),$$

where $\overline{e_o}$ denotes the complex conjugate of e_o. This formulation allows asymmetric relations to be represented via the imaginary part of the embeddings, and symmetry to emerge naturally when imaginary components vanish. Training again uses a margin ranking loss between positive and negative triples.

3.3.4 SimplE

SimplE Kazemi and Poole [2018] is a symmetric variant of the bilinear family, designed to fix the under-constrained scoring of DistMult. Each entity has two embeddings: one when acting as a subject (e_h) and one as an object (e_t). Each relation r also has an inverse counterpart r^{-1}. The scoring function averages both directions:

$$f_{\text{SimplE}}(s, r, o) = \frac{1}{2}\left(\langle e_h(s), r, e_t(o)\rangle + \langle e_h(o), r^{-1}, e_t(s)\rangle\right),$$

where $\langle a, b, c\rangle = \sum_k a_k b_k c_k$ is the tri-linear dot product. This explicitly enforces the duality between each relation and its inverse, making SimplE particularly suitable for reversible relational structures.

3.3.5 Training Procedure

All models are trained using a stochastic margin-based ranking loss:

$$\mathcal{L} = \max(0, \gamma + f(s, r, o^-) - f(s, r, o^+)),$$

where (s, r, o^+) is a true triple and (s, r, o^-) is a negative one generated by corrupting either s or o. In practice, we used 1–5 negative samples per positive triple, optimized with Adam and Xavier initialization. For RotatE, entity embeddings are renormalized to unit modulus and relation phases are wrapped within $[-\pi, \pi]$ after every step.

3.3.6 Conceptual and Empirical Comparison

TransE and RotatE rely on geometric transformations (additive and multiplicative, respectively), while ComplEx and SimplE use multiplicative bilinear interactions. In the context of the reversal curse, SimplE and RotatE should in principle handle inverse relations more naturally, thanks to their explicit modeling of r^{-1} and phase inversions.

Empirical observations. When evaluated on our synthetic reversal datasets, we observe a clear hierarchy of performance: for a single relation, both **ComplEx** and **TransE** reach perfect generalization (MRR ≈ 1.0), while **RotatE** performs moderately (MRR ≈ 0.08) and **SimplE** fails to generalize (MRR ≈ 0.003). As the number of relations increases to six, **ComplEx** maintains perfect generalization (MRR ≈ 1.0), but **TransE** and **RotatE** both degrade to around 0.2, and **SimplE** remains near zero. This suggests that ComplEx can leverage its complex-valued bilinear structure to encode and invert relations even in multi-relation settings, whereas models relying on purely geometric transformations struggle to maintain consistency across multiple relational mappings.

We believe that studying these toy architectures can provide valuable insights into how the *reversal curse* may be overcome in more complex models, such as autoregressive language models. In particular, each of these architectures represents a distinct hypothesis about how relational structure can be encoded geometrically: through translation (TransE), rotation (RotatE), complex-valued bilinear interactions (ComplEx), or symmetric factorization (SimplE).

By analyzing their limitations and inductive biases, we can better understand which geometric principles favor the emergence of invertible relations. In the following section, we reinterpret the TransE architecture through the lens of our transformer-based model, and show how similar mechanisms can arise internally within attention and value–output projection layers.

3.4 Grokking the reversal curse

As we have previously discussed, the autoregressive objective is considered to be, at the moment, the main cause of the reversal curse in LLMs [Lv et al., 2024, Zhu et al., 2024].

A new study, however, (Wang and Sun [2025]) has led to reconsiderations of this hypothesis. The authors have indeed showed that transformers can effectively break the reversal curse, without any changes to the architecture or the objective function.

The authors experimentally find that generalization is achieved spontaneously in synthetic settings, which is something that has previously been unnoticed by the many studies that attempted to study the phenomenon in the wild.
It seems natural to think, then, that the reversal curse is caused by something that only happens "in the wild", since the LLMs are not affected by the problem in simpler settings.
Further investigation demonstrates that, indeed, the "binding problem", as called in the paper, is what afflicts LLMs in more realistic scenarios.
To explore this, we first introduce the synthetic setting where the LLMs can break the reversal curse, and then deduce the differences and complications that can arise in different contexts.

3.4.1 Experimental Setup

In this synthetic setting, our dataset is comprised of token triples, including a subject entity, a relation and an object entity $(e_1\ r\ e_2)$.
For each relation r, we have an inverse relation r^{-1} that helps encode the same fact on the other direction (for example, if e_1 and e_2 are linked via the relation r, this fact will be present twice in the dataset in the forms $(e_1\ r\ e_2)$ and $(e_2\ r^{-1}\ e_1)$).
We then split the data into a training and test set, so that the training set contains a portion of the data with both directions and another, smaller, percentage of the data with only one direction per each fact (this part will be called the atomic set).
In this way, the model will have the possibility, during training, to learn the bidirectionality of the relations, since the non-atomic entries appear twice in the training dataset, and the model has plenty of data to infer their symmetric nature.
On the other hand, the "inverse facts" of the atomic entries serve as test tuples, to assess the model generalization capabilities.

The authors emphasize how the model, that manages to break the reversal curse in this toy setting, struggles to achieve the same performances when we allow the entities to span several tokens, instead of just one.
A natural conclusion is that LLMs can reverse facts at the concept level, but face issues when the input is presented in surface form.
We are interested in exploring the factors that allow the model to generalize to the unseen directions (in the basic scenario we described above).
In particular, we ought to use techniques from mechanistic interpretability to uncover the circuit used by the model to retrieve the correct answers.

3.4.2 Static Mechanistic Interpretability

Our first experiments apply standard mechanistic interpretability techniques in a "static" way, analyzing model checkpoints during training rather than examining how the training process directly shapes model development.
This enables us to examine how the training procedure steers the model toward generalization in an indirect way, by investigating the differences in the model checkpoints.

We begin by examining the model in the later stages of training to identify which components are essential for producing correct answers. This step is central to mechanistic interpretability, as it reduces the complexity of the analysis by focusing on specific layers and architectural elements rather than the model as a whole. Our goal is to isolate a minimal circuit that explains how and where the correct answer first emerges. Naturally, as is typical with transformers and other deep learning architectures, the signal is not confined to a single component. Instead, the answer is progressively refined across layers, with each step contributing incrementally toward the correct output.

Minimal Training Circuit

Our analysis of the training data reveals that the model relies on a very limited depth: the retrieval mechanism responsible for identifying most correct answers requires only the first layer, which already suffices to rank it as the top-1 logit.
To demonstrate this, we patch the final output of a given layer directly into the last layer, before the unembedding and softmax are applied. We repeat this procedure for each layer in turn, allowing us to assess the model's performance when the output from that layer, rather than the final layer, is used.
The results are presented in table 3.4:

Table 3.4: Model performance when patching the output of each layer into the final layer before unembedding.

Layer patched	Top-1 Accuracy
0	0.939
1	0.941
2	0.999
3	0.999
4	0.999
5	0.999
6	0.999
7	0.999
8	0.999
9	0.999
10	0.999
11	0.999

The model is able to correctly predict the vast majority of training tuples immediately after the first layer (3,661 out of 3,900), with further refinement occurring in layers 1 and 2. Subsequent layers beyond layer 2 do not significantly improve training accuracy.
The model's apparent shallowness is likely a consequence of the task's relative simplicity. The sequences are very short, and the limited number of entities and relations further contribute to this effect.

Motivated by these results, we examine the first layer to identify where the knowledge of the training data is stored.
In general, large language models tend to store information in the MLP layers, while the attention mechanism primarily propagates this information across tokens, thereby enriching contextual representations [Geva et al., 2021, Meng et al., 2022].
In our case, we expect the first MLP layer to retain the associations between entity pairs, with attention serving to integrate the information about the subject entity and the relation.
To validate this hypothesis, we employ the causal tracing technique introduced by Meng et al. [2022], in which correct representations are patched into a corrupted input. An observed increase in the correct answer's score provides strong evidence that the corresponding module is responsible for storing the fact.

We construct a corrupted training set by replacing the subject token in each tuple with a random token. Correct representations are then patched into the model running on this corrupted data to assess whether they restore the model's retrieval capability. We repeat the experiment with a second corrupted dataset, this time replacing the relation token with a random (but valid) relation token. The results (average probability of the correct token) are reported in Tables 3.5. and 3.6
It is important to note that, because the output of the attention layer directly feeds into the MLP, we cannot patch only the attention activations. The corrupted MLP activations must also be patched; otherwise, the observed score would be confounded by the indirect beneficial effect on the MLP. Indeed, naive attention patching without this correction yields an inflated score of 0.99.

Table 3.5: Causal tracing corrupting subject token

Layer	Attn Acc	Attn Avg	MLP Acc	MLP Avg
0	0.729	0.710	0.107	0.104
1	0.007	0.005	0.001	0.001
2	0.002	0.001	0.061	0.058
3	0.001	0.001	0.001	0.001
4	0.001	0.001	0.001	0.001
5	0.001	0.001	0.001	0.001
6	0.001	0.001	0.002	0.002
7	0.001	0.001	0.001	0.001
8	0.001	0.001	0.001	0.001
9	0.001	0.001	0.001	0.001
10	0.002	0.001	0.001	0.001
11	0.001	0.001	0.001	0.001

Surprisingly, the causal tracing experiments reveal a different picture than initially expected. When we corrupt the subject token, the attention mechanism achieves a higher score than the MLP.
For example, when corrupting the subject token of a tuple $(s, r, o) \rightarrow (s^*, r, o)$ but patching in the correct attention activations, the model is able to recover an average of 70% of the correct answers. This indicates that the attention mechanism itself is, at least partially, responsible for storing the training associations.
On the other hand, we find that the first MLP layer is able to recover nearly 70% of the correct answers when patched into inputs where the relation token is corrupted. Partial recovery is also observed throughout the rest of the model, extending even to the final layers. This is a noteworthy finding, as it sheds light on the nature of the retrieval process. While the model's

Table 3.6: Causal tracing corrupting relation token

Layer	Attn Acc	Attn Avg	MLP Acc	MLP Avg
0	0.602	0.594	0.678	0.665
1	0.317	0.308	0.164	0.162
2	0.164	0.162	0.246	0.244
3	0.164	0.162	0.164	0.162
4	0.164	0.162	0.164	0.162
5	0.164	0.162	0.164	0.162
6	0.164	0.162	0.164	0.163
7	0.164	0.162	0.164	0.163
8	0.164	0.162	0.164	0.162
9	0.164	0.162	0.164	0.162
10	0.164	0.163	0.164	0.163
11	0.165	0.163	0.164	0.163

knowledge of entity associations (i.e., which entities are "connected" to which) is more localized, the signal corresponding to the correct relation, one that ultimately produces the right answer, appears to be both more robust to corruption and more diffusely stored across the network.
Although this result appears promising, we maintain that, at least in the context of the training circuit, the first-layer MLP and subsequent modules primarily serve to amplify the signal and boost the probability of producing the correct answer.
Nonetheless, our findings suggest that for the majority of the training data, the answer is fundamentally constructed within the attention mechanism of the first layer.

We further validate this finding by applying zero ablation to the MLP output of the first layer, effectively removing the contribution of the MLP module and measuring the impact on training accuracy. The model still achieves 95% accuracy on the training data under this ablation. Together with our earlier result on the model's shallowness, this supports the hypothesis that training knowledge is primarily stored in the attention mechanism.

After reducing the model's complexity by identifying the components most critical for accurate training retrieval, we proceed to a more detailed analysis of the attention mechanism. Specifically, we investigate the attention patterns of individual heads.
Since the sequences are very short, each head can attend either to the subject token or to the relation token. Our results show that for all tuples sharing the same relation ID, the allocation of heads attending to the subject versus the relation remains invariant.
In other words, substituting the subject token while keeping the relation fixed does not alter the positions from which the attention heads extract information.
This indicates that all tuples associated with the same relation type are processed in a structurally consistent manner.

An example is showed in table 3.7, where we compute the average attention weights for all the samples that involve the first relation ID.

We also find that "coupled" relations, where one relation is the inverse of the other (e.g., $(e_1, r, e_2) \iff (e_2, r^{-1}, e_1)$), tend to display more similar patterns. In fact, the cosine similarity between patterns of such coupled relations can reach up to 90%, whereas the similarity between two random relations averages around 70%.

Moreover, as we will discuss in the dynamic interpretability section, this similarity increases during training: starting at random levels, it gradually rises as the model learns. This phenomenon

Table 3.7: Average attention weights from the subject and relation tokens to the last token across heads in Layer 0.

Head	Subject	Relation
Head 0	0.9626	0.0374
Head 1	0.0031	0.9969
Head 2	0.9906	0.0094
Head 3	0.9992	0.0003
Head 4	0.0004	0.9996
Head 5	1.0000	0.0000
Head 6	0.9947	0.0053
Head 7	0.9915	0.0085
Head 8	0.9969	0.0006
Head 9	0.9983	0.0017
Head 10	0.0000	1.0000
Head 11	0.0114	0.9886

is not isolated; it is consistently observed across different model components: embeddings, activations, and attention head patterns. Interestingly, tracking this evolution throughout training shows that the similarity between coupled relations builds up in parallel with improvements in test accuracy and generalization.

An example of this effect is illustrated in Figure 3.1, which depicts the pairwise similarity between the embeddings of the relation IDs.

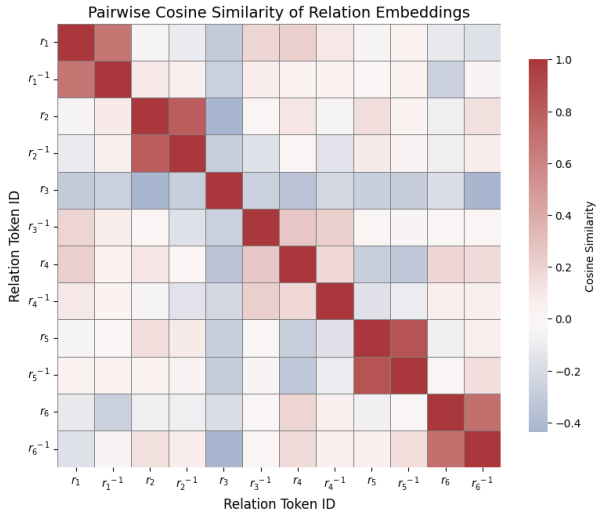


Figure 3.1: Pairwise relation embedding similarity

The attention mechanism can be conceptualized as comprising two distinct circuits.
The Query–Key circuit is responsible for determining where to attend within the sequence, i.e., identifying the most relevant tokens for transmitting information and contextual cues.
In contrast, the Value–Object circuit governs what information is actually propagated to the subsequent layer. Having first analyzed the behavior of the Query–Key circuit (thereby addressing the question of where information is retrieved), we now turn our attention to the Value–Object circuit, which dictates what is written to the residual stream.

We observe that the model's combined W_{VO} matrix encodes information about entity associations. Specifically, since each entity is connected to k other entities (where k corresponds to

the number of relation types, excluding inverses), we find that, for each entity, the majority of its "neighbors" (i.e., the entities with which it is associated) can be recovered through the W_{VO} mechanism. We indeed find that an average of 70% of the neighbors can be retrieved for each tuple in the training set, just using the W_{VO} module and the embedding of the subject entity, which we call s.

Formally, we compute

$$a_i = s \cdot W_{VO_i}, \quad i = 1, \ldots, h.$$

We then average the resulting vectors across all heads:

$$a = \frac{1}{h} \sum_{i=1}^{h} a_i.$$

Where we call h the number of heads of the model.
The resulting vector, corresponding to what the attention mechanism would produce if all heads attended to the subject token, is strongly aligned with the neighbors of the subject token, i.e., the embeddings of the k object tokens associated with the k relations. We quantify this using the logit-lens technique, treating the vector as if it were the final token activation of the last layer. Specifically, we unembed the vector via the unembedding matrix and evaluate how many of its top-k logits correspond to the k neighbors.

As mentioned previously, the attention patterns (which we can think of as binary vectors of length h, where 1 indicates that the head is responsible for transferring information from the subject token to the last token, and 0 if it is from the relation token instead) are nearly identical for all sequences that share the same relation ID.

If, unlike previously, where we summed all the vectors generated by multiplying the embedding s with the W_{VO} matrix, we instead sum only the contributions corresponding to the attention pattern for a specific relation, we obtain a resulting vector more aligned with a single neighbor rather than encoding all of them. In other words, if previously most neighbors were in the top-k logits encoded by the vector a, selecting the correct subset of contributions a_i (matching the pattern of relation r) can bring the correct neighbor o as the top-1 logit.

$$a_i = s \cdot W_{VO_i}, \quad i = 1, \ldots, h.$$

We now only consider the vectors matching the pattern p of relation r:

$$a_r = \frac{1}{|p|} \sum_{i \in p} a_i.$$

When we unembed the vector a_r for each (s, r) pair in the training set, we obtain a top-1 accuracy of approximately 65%, compared to only 20% when using the naive summation. This result is expected, since the procedure effectively simulates how the attention mechanism operates. What is more interesting, however, is that, given the simplicity of the task, the model converges on a minimal and efficient strategy for selecting the correct token: it encodes information about all candidate objects, and the attention mechanism acts as a selector that extracts the appropriate subset corresponding to the relation. Remarkably, this mechanism, based solely on combinations of the value vectors from the subject token, manages to capture a substantial fraction of the training associations.

Equally striking is the observation that, because all tuples involving a given relation r_i share the same attention pattern p_i, the corresponding associations are consistently stored within the

same subset of the W_{VO} matrix. In effect, this suggests that each relation induces a dedicated linear map linking subject entities to their associated objects.

We have thus clarified more about the two mechanisms that compose attention: the VO circuit contains information about the neighbors of a given subject entity, while the QK circuit helps select the "right" neighbor depending on the relation token.

We now turn to the test set to examine whether these findings extend beyond the training tuples. Specifically, we ask: is this same circuit responsible for the model's ability to generalize, and does it play a role in overcoming the reversal curse? These questions will be addressed in the following section.

3.5 Symmetry and the Emergence of Generalization

We demonstrated that the attention mechanism selects the correct neighbor based on the given relation, amplifying it relative to incorrect ones, with MLPs providing additional refinement. We found that the same attention circuits can retrieve neighbors even during testing, including those the model has never encountered during training.

To clarify this finding, recall that given k distinct relations (excluding inverses), each entity connects to exactly k neighbors. While some entities have all their neighbors present in the training set, others have some connections reserved for testing, making this relational information unavailable during learning. Despite this limitation, the model consistently identifies these unseen entities among the most likely tokens at inference time.

To quantify this effect, we conducted a systematic evaluation measuring the proportion of cases where true neighbors appear among the top-ranked predictions. We first constructed neighbor sets for each entity by aggregating all relations from both training and test data, such that each subject has exactly k neighbors corresponding to the k distinct relations in our dataset. We then computed the fraction of examples where at least one true neighbor appears among the top-10 predicted tokens, using logits extracted from the residual stream after the first attention layer.

The results demonstrate remarkable performance: the model achieves perfect neighbor retrieval on training data and maintains near-perfect accuracy on test data (99.8%), indicating robust generalization across the knowledge graph structure. To further isolate the model's ability to predict truly unseen connections, we partitioned neighbors into two categories: seen neighbors that appeared during training and unseen neighbors reserved exclusively for testing. Even for these completely novel connections, the model exhibits substantial predictive capability, identifying unseen neighbors within the top-10 predictions in 86.0% of cases. This strong success rate for previously unobserved connections provides compelling evidence that the attention mechanism learns generalizable relational patterns rather than merely memorizing training associations.

These results confirm that the first attention layer is the primary source of the model's generalization capabilities: while it achieves perfect retrieval on training data, it continues to produce high-confidence predictions for unseen examples, even before later layers refine the output.

From retrieval to structure. In order to understand where these generalization abilities originate, we now analyze the structural properties of the learned attention parameters. Specifically, we focus on the composite matrix:

$$S = \sum_h W_{V_h} W_{O_h},$$

which we have previously shown to be sufficient to recover the correct neighbors of each entity. This finding suggests that the key mechanism underlying generalization is not distributed across the full transformer architecture, but instead resides in the internal geometry induced by S between entity representations.

Symmetry as the driving factor. We will show that during training, the matrix S becomes progressively more symmetric, with its symmetric component

$$S_{\text{sym}} = \tfrac{1}{2}(S + S^\top)$$

growing in norm relative to its skew component

$$S_{\text{skew}} = \tfrac{1}{2}(S - S^\top).$$

This gradual increase in symmetry corresponds directly to the onset of generalization: as soon as $\|S_{\text{sym}}\|$ overtakes $\|S_{\text{skew}}\|$, the model begins to produce correct reversed associations that were never observed in training.

This phenomenon can be understood as a structural regularization effect. During training on bidirectional data, the gradient of S has an inherent symmetric bias: each pair of reciprocal relations (s, r, o) and (o, r^{-1}, s) produces complementary updates that amplify the symmetric component and suppress the skew one. Even when S_{skew} remains non-zero, the optimization process consistently favors solutions where $S \approx S^\top$, thereby enforcing a symmetric mapping between subjects and objects.

In the following subsections, we analyze this mechanism more formally, first quantifying the gradient bias induced by the dataset structure and then empirically verifying the evolution of S_{sym} and S_{skew} throughout training. We show that the emergence of symmetry in S is the fundamental driver of the model's ability to generalize beyond its observed knowledge graph.

3.5.1 Symmetry in the Attention Operator and Its Role in Generalization

Motivation. We have shown that the matrix

$$S = \sum_h W_{V_h} W_{O_h}$$

is sufficient to retrieve the correct neighbors of each entity, both in training and testing. We now turn to understanding how this structure evolves during optimization and why it leads to generalization on unseen inverse pairs. In particular, we analyze the gradient of the loss with respect to S to reveal an intrinsic bias toward symmetry that emerges directly from the training signal.

Gradient of the bilinear model. We consider the simplified bilinear setting $\ell_o = e_x^\top S e_o$, where e_x and e_o are the subject and candidate embeddings. The model's predictive distribution is

$$p(o \mid x) = \frac{\exp(\ell_o)}{\sum_{k \in E} \exp(\ell_k)},$$

and the per-example cross-entropy loss is

$$\mathcal{L}(x, y) = -\log p(y \mid x) = -e_x^\top S e_y + \log \sum_{k \in E} \exp\Big(e_x^\top S e_k\Big).$$

Differentiating the loss yields

$$\frac{\partial \mathcal{L}}{\partial S} = \sum_{o \in E} (p(o \mid x) - \mathbf{1}\{o = y\})\, e_x e_o^\top = e_x m(x)^\top - e_x e_y^\top,$$

where we define the softmax mixture

$$m(x) = \sum_{o \in E} p(o \mid x)\, e_o.$$

Thus, the gradient takes the compact form

$$\boxed{\frac{\partial \mathcal{L}}{\partial S} = -e_x e_y^\top + e_x m(x)^\top.}$$

Symmetry of the training signal. At the beginning of training, the model distribution $p(o \mid x)$ is nearly uniform, so $m(x)$ is close to the global mean of embeddings. In this regime, the gradient is dominated by the term $-e_x e_y^\top$, corresponding to a rank-1 update that pushes the parameter matrix to increase the compatibility between subject x and target y. When both directions $(x \to y)$ and $(y \to x)$ are present in the training data, their combined update is

$$-(e_x e_y^\top + e_y e_x^\top),$$

which lies entirely in the symmetric subspace. If only one direction is present ("atomic" edges), the gradient splits evenly between symmetric and skew parts. Since our datasets contain many bidirectional pairs and fewer atomic ones, the cumulative training signal projects more strongly onto the symmetric subspace.

As training progresses, the second term $e_x m(x)^\top$ acts as a soft correction that further amplifies symmetry: the mixture $m(x)$ itself becomes correlated with true neighbors, and empirically its symmetric component dominates ($\|M_{\text{sym}}\|_F / \|M_{\text{skew}}\|_F \approx 2.0$). Overall, these two effects, paired edges and softmax mixtures, jointly bias gradient descent toward increasingly symmetric updates.

Empirical symmetry and neighbour retrieval. At convergence, we decompose $S = S_{\text{sym}} + S_{\text{skew}}$ and evaluate the induced scores $s(b \mid a) = e_b^\top S e_a$. The symmetric kernel $K(a, b) = e_b^\top S_{\text{sym}} e_a$ is already sufficient to recover the full neighborhood structure: on both train and test sets, K achieves Hit@10 = 1.0 and Recall@10 0.93. Adding the skew term slightly perturbs the ranking, reducing Recall@10 to 0.8, but Hit@10 remains nearly perfect. Hence, the symmetric component S_{sym} fully captures the relational graph, including unseen test pairs.

Coherence and optimizer bias. To understand why symmetric updates dominate, we computed the coherence of per-edge gradients,

$$\text{coh} = \frac{\|\sum_i g_i\|}{\sum_i \|g_i\|}.$$

Symmetric gradients exhibit high coherence ($\text{coh}_{\text{sym}} \approx 0.06$, well above a random baseline), while skew gradients show destructive interference ($\text{coh}_{\text{skew}} < 0.02$). The ratio $\|G_{\text{sym}}\| / \|G_{\text{skew}}\|$ remains consistently above one (1.1–1.3 throughout training), whereas randomized control data fluctuate around unity. These results indicate that only symmetric gradients add coherently across examples, producing a stable optimization signal, while skew gradients cancel out. Consequently, the optimizer implicitly reinforces S_{sym}, causing S to become progressively more symmetric over time.

To conclude, training on bidirectional data naturally induces an intrinsic bias toward symmetry in the attention operator. As the symmetric component S_{sym} grows relative to S_{skew}, the model acquires the ability to generalize to unseen inverse relations. This emergent symmetry provides a direct and interpretable link between gradient dynamics, dataset structure, and relational inference.

3.5.2 Embedding Geometry and Relational Structure

Motivation. In the previous subsection, we demonstrated that generalization arises primarily from the emergence of symmetry in the attention operator S, which acts as a relational kernel mapping subject embeddings to their correct neighbors. However, this mechanism presupposes that the underlying entity embeddings provide a stable and meaningful basis for such mappings to operate on. It is therefore natural to ask whether the embeddings themselves encode any interpretable relational geometry, and to what extent this geometry resembles that learned by classical knowledge-graph models such as TransE, RotatE, ComplEx, or SimplE. Studying the embedding space allows us to compare the inductive biases of the transformer to those of explicit relational networks, and to understand whether the model develops analogous compositional structures.

First-order (TransE-style) structure. We first tested for a *translational* geometry of the form $e_o \approx e_s + t_r$, analogous to the inductive bias of TransE. For each relation r, we computed the mean translation vector

$$t_r = \mathbb{E}_{(s,r,o)\in\text{train}}[\, e_o - e_s \,],$$

and evaluated whether the predicted object $\hat{e}_o = e_s + t_r$ ranked the true o highly among all entities. Performance was poor (MRR = 0.017, Hits@1 = 0.000, Hits@10 = 0.040), showing that the embeddings do not follow a consistent global translation rule. Thus, unlike TransE, the transformer does not represent relations as fixed displacement vectors in embedding space.

Analogy (3CosAdd) structure. We next tested for word2vec-style analogical regularities using the standard 3CosAdd rule. Given two training pairs (s_1, r, o_1) and (s_2, r, o_2), we predicted o_2 as

$$\hat{e}_{o_2} = \text{norm}(e_{s_2} + (e_{o_1} - e_{s_1})).$$

Accuracy was effectively zero, confirming that linear analogical composition is absent from the embedding space. Relations are not encoded as simple additive directions.

Local directional biases. Although global translations fail, each relation still induces a weak directional bias: its mean direction t_r separates true pairs from random negatives with AUCs in the range 0.66–0.75. Hence, embeddings exhibit local anisotropies reflecting relational tendencies, but these directions vary across relations and are not globally compositional.

First- vs. second-order relational similarity. To characterize the geometry more globally, we compared embedding similarities with graph-based relational similarities. For 5,000 random entity pairs, we computed (i) cosine and Jaccard similarities over direct neighbor sets (first-order) and (ii) over two-hop co-neighbor contexts. Embedding cosine correlated only weakly with first-order measures ($r \approx 0.18$), but more strongly with second-order measures ($r \approx 0.25$). This indicates that the embedding space primarily encodes *distributional* rather than direct relational information: entities with similar neighborhoods—not necessarily directly connected—tend to cluster together.

Gradient-based interpretation. This behaviour can be understood from the structure of the gradient with respect to entity embeddings. For a training triple (a, r, b) with cross-entropy loss $\mathcal{L}(a, r, b) = -\log p_\theta(b \mid a, r)$, the gradient on the subject embedding can be expressed as

$$\nabla_{e_a}\mathcal{L} = \left(\frac{\partial h_t}{\partial e_a}\right)^\top (W_U^\top p_\theta(\cdot \mid a, r) - e_b),$$

where W_U is the unembedding matrix and $\frac{\partial h_t}{\partial e_a}$ is the Jacobian describing how e_a influences the output state through attention. Because GPT-style models use tied input–output embeddings ($W_U = E^\top$), the term $W_U^\top p_\theta(\cdot \mid a, r)$ represents the model's current predicted mean embedding,

while e_b is the target embedding. The update thus moves e_a to align the model's predicted mean with e_b, aggregating over all its relational contexts. Entities that share similar output distributions therefore receive correlated updates, naturally forming clusters corresponding to similar relational roles.

Empirical validation. Linear probes trained on these embeddings confirmed this interpretation. Raw embeddings achieved near-random discrimination for 1-hop neighbor prediction (AUC ≈ 0.53), but much higher accuracy for 2-hop structures (AUC 0.88–0.92). Furthermore, directly optimizing embeddings via a contrastive (InfoNCE) loss on observed edges produced a configuration capable of near-perfect neighborhood retrieval (MRR = 0.39, Hits@10 = 0.97). Hence, the embedding space is theoretically expressive enough to encode the full relational graph, but the pretrained language model does not spontaneously discover a perfectly local or translational geometry. Instead, embeddings form a soft distributional scaffold that supports the symmetric attention operator S, on top of which relational inference and generalization emerge.

Summary. The embeddings learned by the transformer differ qualitatively from those of explicit relational models such as TransE or RotatE. They lack global linearity but capture higher-order relational similarity, providing a distributed basis that enables S to learn a symmetric relational mapping. Rather than encoding relations directly in the embeddings, the model relies on the attention mechanism to construct relational structure dynamically, with embeddings serving as stable anchors in a geometry shaped by contextual co-occurrence.

3.5.3 Bilinear Training

In this work, we progressively simplified the transformer architecture by ablating or freezing specific submodules, with the goal of identifying the smallest configuration that is still capable of breaking the reversal curse.
We found that, on simple synthetic datasets, the model discovers a remarkably minimal circuit that generalizes, relying only on the embedding layers and a single attention layer.
Moreover, due to the simplicity of the training data, composed only of subject, relation, and mask tokens, we observed that the model effectively relies on the subject embedding alone to produce meaningful outputs, both during training and testing.

In particular, we confirmed that, under several simplifying assumptions, the logits produced by the model are approximately proportional to:

$$\text{logits} = e_s^\top S E^\top,$$

and consequently, the score assigned to a candidate object token o can be expressed as:

$$\text{score}(o) = e_s^\top S e_o,$$

where, as a reminder, we define:

$$S = \sum_h W_{V_h} W_{O_h}.$$

We noticed that, in practice, the model behaves almost identically to a simple bilinear model, with the matrix S acting as a bridge between subject and object embeddings.
Motivated by this observation, we implemented a toy bilinear model that explicitly replicates the internal mechanism the transformer appears to learn, in order to isolate the *smallest architecture that breaks the reversal curse* while remaining consistent with transformer dynamics.

The model consists of an embedding layer of size $V \times d$, where V denotes the vocabulary size, including all entities and relations—and a learnable parameter $S \in \mathbb{R}^{d \times d}$, representing the value–output circuit within the transformer.
As expected, we found that this simplified bilinear model converges and generalizes perfectly on the test set, closely matching the behavior of the full transformer.

We then analyzed the gradient of the loss with respect to S, since in this setting S is an explicit model parameter. This makes gradient analysis straightforward, unlike in the transformer, where S is defined implicitly as a product of learned parameters W_V and W_O, and computing its effective gradient requires combining their respective gradients through the product rule.

We now estimate the expected symmetry of the gradient with respect to S in the toy bilinear model, given the structure of our training dataset. Recall that for a single training triple (s, r, o), under the early-training approximation where the model's output distribution is nearly uniform, the gradient takes the form:

$$\nabla_S \mathcal{L}(s, r, o) \propto e_s e_o^\top,$$

that is, the outer product between the subject and object embeddings.

Paired examples. For every bidirectional pair (s, r, o) and its inverse (o, r^{-1}, s), the total gradient contribution is:

$$G_{\text{pair}} = e_s e_o^\top + e_o e_s^\top.$$

It follows immediately that $G_{\text{pair}}^{\top} = G_{\text{pair}}$, hence this contribution is *purely symmetric* and has no skew component. Assuming approximately orthogonal and unit-normalized entity embeddings, we have:

$$\|e_s e_o^{\top}\|_F^2 = 1, \quad \langle e_s e_o^{\top}, e_o e_s^{\top} \rangle = 0,$$

and therefore:

$$\|G_{\text{pair}}\|_F^2 = 1 + 1 = 2.$$

Single (atomic) examples. For single-direction triples (no inverse counterpart), we have:

$$G_{\text{single}} = e_s e_o^{\top}.$$

Decomposing it into symmetric and skew components:

$$G_{\text{single}}^{\text{sym}} = \tfrac{1}{2}(e_s e_o^{\top} + e_o e_s^{\top}), \quad G_{\text{single}}^{\text{skew}} = \tfrac{1}{2}(e_s e_o^{\top} - e_o e_s^{\top}).$$

Under the same orthogonality assumption, the Frobenius norms of both components are equal:

$$\|G_{\text{single}}^{\text{sym}}\|_F^2 = \|G_{\text{single}}^{\text{skew}}\|_F^2 = \tfrac{1}{2}.$$

Aggregate gradient over the dataset. Suppose our dataset contains n paired examples (i.e., $2n$ directions) and $0.6n$ single-direction examples. The total symmetric and skew contributions are therefore:

$$\|G_{\text{sym}}\|_F^2 = n \cdot 2 + 0.6n \cdot \tfrac{1}{2} = 2.3n,$$
$$\|G_{\text{skew}}\|_F^2 = n \cdot 0 + 0.6n \cdot \tfrac{1}{2} = 0.3n.$$

Taking the Frobenius norms:

$$\|G_{\text{sym}}\|_F = \sqrt{2.3n}, \qquad \|G_{\text{skew}}\|_F = \sqrt{0.3n}.$$

Symmetry share. The expected symmetry share is then:

$$g_{\text{sym_share}} = \frac{\|G_{\text{sym}}\|_F}{\|G_{\text{sym}}\|_F + \|G_{\text{skew}}\|_F} = \frac{\sqrt{2.3n}}{\sqrt{2.3n} + \sqrt{0.3n}} = \frac{\sqrt{2.3}}{\sqrt{2.3} + \sqrt{0.3}} \approx 0.735.$$

Interpretation. Thus, under this simplified bilinear analysis, we expect the gradient to be strongly biased toward symmetry ($g_{\text{sym_share}} \approx 0.73$). This confirms that the dataset structure itself, specifically the ratio between paired and single-direction examples, induces a symmetric bias in the gradient, which in turn drives the model toward learning $S \approx S^{\top}$.

Note that the numerous assumptions we have applied to the model, despite allowing us to pinpoint a specific part that is responsible for correct retrieval, might result in a mismatch between the expected results and what we empirically measure.
In particular, our simplifications included disregarding positional encoding, residual connection, MLP and layernorms.
Doing this is fine since they do not alter the behavior of the model, which still works fairly similarly to the toy bilinear one we described above.

For example, we experimenting ablating positional encodings in the model and the results changed slightly. Similarly we proved that MLPs are actually not essential for generalization.
However, all these components, despite not significantly influencing the core internal mechanisms, can have an effect on the magnitude of some statistics.
This is the reason why, we believe, there is a mismatch between the symmetric gradient ratio in the toy model in contrast to the symmetric gradient ratio in the actual gpt architecture.

Comparison with the transformer results. The gradient symmetry measured in this bilinear setting ($g_{\text{sym}}/(g_{\text{sym}} + g_{\text{skew}}) \approx 0.7$) is remarkably close to the theoretical expectation derived from the dataset structure, where the ratio of bidirectional to atomic edges is known. This confirms that the simplified bilinear dynamics capture the essential training bias toward symmetry. However, when we measure the same ratio in the full transformer, we observe a noticeably lower value, around 0.51–0.52. The difference likely arises from architectural components absent in the toy model, residual streams, layer normalizations, positional encodings, and additional gradient pathways, which diffuse the signal that directly updates S. These results justify the introduction of the bilinear model as a minimal and analytically tractable proxy for understanding how the transformer develops symmetry and, consequently, generalizes to unseen inverse relations.

In the end, deep learning models, especially the most convoluted ones like transformers, can be quite hard to reverse-engineer and simplifications might be needed.

The circuit found in both the transformer and the toy bilinear model is quite different from other neural networks capable of breaking the reversal curse.
For example, we have seen how some simple models, using specific losses that enforce geometry directly, can learn reversal by modifying the embedding layers directly.

These models embed the logic of reversal retrieval in the embedding geometry, finding a structure where subject and object entities are tied algebraically in a high dimensional space.
Similarly, we tried to check whether our bilinear model could be nudged towards finding a similar structure by freezing the S parameter and hardwiring a new set of logits

$$logits = (e_s + e_r)E^T$$

effectively optimizing just the embedding layer.
The motivation is that this is another function a transformer could easily implement and, at the same time, being quite analogous to the transE objective, it should generate a similar structure. Indeed, if it is not clear, we can show that the gradients derived from the cross-entropy loss push e_o towards $e_s + e_r$

$$\mathcal{L} = -\log p_o = -z_o + \log \sum_{v=1}^{V} e^{z_v}.$$

At the beginning of training, the logits are small and nearly uniform, so $p_v \approx \frac{1}{V}$. In this regime, the gradient of the loss with respect to the logits is:

$$\frac{\partial \mathcal{L}}{\partial z_v} = p_v - \mathbf{1}_{[v=o]} \approx \tfrac{1}{V} - \mathbf{1}_{[v=o]}.$$

Hence, for the correct token o, we have a large negative gradient, while for all other tokens the gradient is slightly positive. This means the update pushes the embedding of the correct token e_o *towards* the direction of $(e_s + e_r)$, and conversely pulls $(e_s + e_r)$ towards e_o. Thus, even from random initialization, the model starts aligning $e_o \approx e_s + e_r$.

The derivative of the loss w.r.t. each logit is:

$$\frac{\partial \mathcal{L}}{\partial z_v} = p_v - \mathbf{1}_{[v=o]}.$$

Since $z_v = (e_s + e_r) \cdot E_v$, where E_v is the embedding vector of token v, we obtain:

$$\frac{\partial \mathcal{L}}{\partial e_s} = \frac{\partial \mathcal{L}}{\partial e_r} = \sum_{v=1}^{V} (p_v - \mathbf{1}_{[v=o]}) E_v = (p - y)^\top E,$$

where y is the one-hot vector for the correct object token.

Each embedding vector E_v contributes linearly to its logit:

$$\frac{\partial \mathcal{L}}{\partial E_v} = (p_v - \mathbf{1}_{[v=o]})(e_s + e_r).$$

In matrix form, this becomes:

$$\frac{\partial \mathcal{L}}{\partial E} = (p - y) \otimes (e_s + e_r) = (p - y)(e_s + e_r)^\top.$$

Interpretation

At the start of training ($p \approx \frac{1}{V}$), the gradients point roughly in the direction of:

$$\nabla_E \mathcal{L} \propto -y(e_s + e_r)^\top,$$

which strengthens the inner product between e_o and $(e_s + e_r)$. During training, as we expected, this leads to the emergent geometric relation:

$$e_o \approx e_s + e_r$$

Due to the structure of our datasets, we also observe that the embedding of the relation r (for simplicity, we are using just one relation and its inverse in this setting) and the inverse r_{inv} grow to be almost completely negatively aligned

$$e_r \approx -e_{r_{inv}}$$

We indeed observe a monotonically decreasing trend in the measured cosine similarity between e_r and $e_{r_{inv}}$, that starts around zero (expected value for random data) and surpass -0.95, meaning that effectively, the model learned the two relations have opposite meaning and therefore, directions in the embedding space.

This is because, due to the bidirectionality in the dataset, where most of the facts are non-atomic and contain both directions, the model, has to optimize simultaneously, for each triple (s,r,o):

$$e_o \approx e_s + e_r$$

$$e_s \approx e_o + e_{r_{inv}}$$

combining the two, we obtain

$$e_s \approx e_s + e_r + e_{r_{inv}}$$

$$e_r \approx -e_{r_{inv}}$$

This is an interesting study case because it shows an alternative circuit the model could be able to reproduce, although it is apparently not incentivised to, during training.
Indeed, the transformer would just need to set the value weights as essentially identity mappings and set the attention weights as 0.5 for both subject and object.

$$\alpha_s \approx \alpha_r \approx 0.5$$

$$v_s \approx e_s,\ v_r \approx e_r$$

This shows how autoregressive GPT-like models can, theoretically, learn two algorithms to break the reversal curse.
Although both are theoretically possible for the model to learn, we empirically observe the model always opting for storing the actual mapping in the attention layer, rather than modifying the embedding geometry.
We hypothesize this happens for a bias of transformer models to prefer finding local, contextual transformations in attention or MLP layers, rather than changing global token geometry.

3.5.4 Gap between synthetic and realistic settings

We confirmed that transformers trained on an autoregressive objective can learn to perform the reversal task in a synthetic setting where each sample is a simple (subject, relation, object) tuple.

By analyzing the internal activations, we identified the circuit responsible for this behavior and observed that, in such algebraic and grokking-like settings, the dataset structure itself enforces a symmetric bias in the gradients and, consequently, in the learned model, leading to successful generalization. This observation clarifies why reversal learning is rarely seen in natural language: the data used to train large language models does not exhibit such closed and symmetric patterns. We also found that test performance decreases as the number of relations grows, suggesting that, in more realistic settings where relations are numerous and often asymmetric, the same mechanism cannot easily emerge. Furthermore, the circuit discovered in the synthetic experiments would be impractical in real-world LLMs, as it effectively dedicates an entire attention layer to symmetry, reducing the flexibility that is essential to these architectures.

At the same time, we showed that transformers are theoretically capable of implementing an alternative algorithm that leverages the geometric structure of the embedding space to overcome the reversal curse, though standard gradient optimization does not incentivize this solution. This method would be more flexible, modifying only a few embeddings instead of an entire layer, but still suboptimal, since embeddings in real autoregressive models must remain free from specific contextual meaning. Ideally, a model should compress each sentence into three latent representations, subject, relation, and object, and then apply a translational geometry similar to TransE, supporting both symmetry and generalization. However, autoregressive transformers lack the inductive bias to build such a structure, unlike autoencoders, which explicitly learn compact latent spaces. Consequently, their latent representations are dense, entangled, and unsuitable for simple vector arithmetic or symbolic reasoning.

Chapter 4

Conclusion

This thesis investigated the reversal curse, the failure of autoregressive transformers to generalize inverse relations, and demonstrated how, under idealized conditions, these models can in fact learn to overcome it. Our findings reveal that when the training data enforces a clear and symmetric structure, transformers can develop internal mechanisms that exhibit genuine generalization, even without explicit supervision.

However, such conditions are rarely met in real-world language data, where asymmetry, contextuality, and semantic richness dominate. In natural text, relations are often not expressed in perfect reciprocal ways and models might rely on superficial statistical correlations rather than structured reasoning. More broadly, the reversal curse exemplifies a key limitation of current large language models: their reliance on statistical correlations rather than abstract reasoning or structured representations. While transformers excel at modeling sequences and capturing the subtle nuances of human language, they are not inherently suited for reasoning within a latent space.

This tension between statistical sequence modeling and conceptual reasoning is one of the central frontiers in modern AI research. We motivate this also by showing how, in the wild, representations of the same concept, in two different "directions" or phrasing are usually in completely different subparts of the model, instead of sharing the same parameters. Bridging this gap may require architectures with stronger inductive biases toward compact, compositional, and interpretable representations, such as JEPA models, autoencoders, or hybrid systems that combine the strengths of both paradigms.

Ultimately, developing such architectures would allow models to *reason* about relations between entities, instead of merely recall them.

Bibliography

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Mikita Balesni, Tomek Korbak, and Owain Evans. The two-hop curse: Llms trained on a-> b, b-> c fail to learn a–> c. *arXiv preprint arXiv:2411.16353*, 2024.

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on" a is b" fail to learn" b is a". *arXiv preprint arXiv:2309.12288*, 2023.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*, 2021.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Mor Geva, Tal Schuster, and Jonathan Berant. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021. URL https://arxiv.org/abs/2012.14913.

Olga Golovneva, Zeyuan Allen-Zhu, Jason Weston, and Sainbayar Sukhbaatar. Reverse training to nurse the reversal curse. *arXiv preprint arXiv:2403.13799*, 2024.

Qingyan Guo, Rui Wang, Junliang Guo, Xu Tan, Jiang Bian, and Yujiu Yang. Mitigating reversal curse in large language models via semantic-aware permutation training. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11453–11464, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.680. URL https://aclanthology.org/2024.findings-acl.680/.

Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

Ouail Kitouni, Niklas Nolte, Diane Bouchacourt, Adina Williams, Mike Rabbat, and Mark Ibrahim. The factorization curse: Which tokens you predict underlie the reversal curse and more. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang,

editors, *Advances in Neural Information Processing Systems*, volume 37, pages 112329–112355. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cbcce87f745072c819204529be843d16-Paper-Conference.pdf.

Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.

Ang Lv, Kaiyi Zhang, Shufang Xie, Quan Tu, Yuhan Chen, Ji-Rong Wen, and Rui Yan. An analysis and mitigation of the reversal curse. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13603–13615, 2024.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2202.05262.

Judea Pearl. Direct and indirect effects. In *Probabilistic and causal inference: the works of Judea Pearl*, pages 373–392. 2022.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Boshi Wang and Huan Sun. Is the reversal curse a binding problem? uncovering limitations of transformers from a basic generalization failure. *arXiv preprint arXiv:2504.01928*, 2025.

Xiao Zhang, Miao Li, and Ji Wu. Co-occurrence is not factual association in language models. *arXiv preprint arXiv:2409.14057*, 2024.

Zheng Zhang. Comprehension without competence: Architectural limits of llms in symbolic computation and reasoning. *arXiv preprint arXiv:2507.10624*, 2025.

Hanlin Zhu, Baihe Huang, Shaolun Zhang, Michael Jordan, Jiantao Jiao, Yuandong Tian, and Stuart Russell. Towards a theoretical understanding of the "reversal curse" via training dynamics. In *Advances in Neural Information Processing Systems*, 2024.