POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Cognitive Aware Incremental Knowledge Update of Large Language Models

Supervisors

Prof. Marco MELLIA

Prof. Johan BOYE

PhD Zied BEN HOUIDI

Candidate

Simone CLEMENTE

October 2024

Abstract

Despite remarkable capabilities, Large language models (LLMs) struggle with incrementally updating knowledge without catastrophic forgetting or indiscriminate learning. In contrast, humans effortlessly integrate new information, detect conflicts with existing beliefs, and selectively update their knowledge. This work introduces a novel paradigm inspired by human brain: Cognitive Aware Incremental Knowledge Update. We implement and evaluate two key components within existing LLM architectures: (1) Inner State Awareness, allowing LLMs to classify new information as novel, familiar, or conflicting; and (2) targeted updates through Differentiated Plasticity, distinguishing between neurons containing previous knowledge (busy) and rarely used neurons (free). Through a series of controlled experiments, we demonstrate the potential benefits of this approach, including improved preservation of prior knowledge during updates, more effective handling of conflicting information, and enhanced ability to target specific knowledge for updates. While challenges remain, particularly in scaling to full-size LLMs and real-world scenarios, our work provides a promising direction for developing more flexible and adaptable language models. In this study, we present a detailed overview of the proposed method, supported by a comprehensive review of the existing literature, a complete description of the experiments, and an in-depth analysis of our findings.

Summary

Introduction

Large Language Models (LLMs) are a category of neural network models trained on immense amounts of text data making them capable to understand, generate, and analyze human-like language through deep learning techniques. Leveraging billions or even trillions of parameters, LLMs can perform a wide range of language-based tasks and provide human-like interactions. They serve as the backbone of various commercial products, such as ChatGPT, Gemini, and Claude, enabling advanced conversational capabilities and problem-solving skills.

While demonstrating remarkable capabilities across various tasks, LLMs face significant challenges in real-world deployment and long-term utility due to the static nature of current training paradigms. Specifically, they present several key limitations in the ability to adapt and learn continuously:

- *Catastrophic forgetting*: incorporating new information in LLMs often leads to erasure of previously learned knowledge. This means that when LLMs are updated with new data, they tend to overwrite or disrupt existing information, rather than seamlessly integrating it.
- *Indiscriminate learning*: LLMs passively accept all training data without the ability to selectively focus on relevant new information while filtering out redundant or conflicting data.
- *Lack of sparsity*: the current training paradigm distributes knowledge uniformly across the entire network, making it challenging to apply targeted updates to specific information without disrupting unrelated knowledge.

On the other hand, humans have the capacity to continuously update their knowledge as they interact with the world. They seamlessly integrate new information, ignore familiar facts, and actively engage in resolving conflicts with existing beliefs before updating their mental models. This cognitive flexibility stems from several key abilities. Humans exhibit (i) selective attention, focusing on novel

or relevant information while filtering out redundant or irrelevant stimuli. They readily (ii) identify inconsistencies between new information and existing knowledge through conflict detection. Humans perform (iii) targeted updates, modifying specific neural pathways related to new information without disrupting unrelated knowledge. Lastly, (iv) human brains maintain adaptive plasticity, favouring stability of well-established knowledge, and showing more flexibility in case of uncertainty.

In this work we introduce a novel human-inspired learning mechanism into existing LLM frameworks, creating a platform for experimenting with targeted learning approaches. Specifically, we test our method in *incremental learning* [1], where we evaluate the ability of learning new information while preserving previously acquired knowledge, and *model editing* [2] which assesses the effectiveness in modifying prior model's beliefs. Our findings pave the way for further research in continual learning, knowledge representation, and cognitive neuroscience-inspired AI.

Proposed Approach

Our technique builds on the concept of cognitively aware updates, where instead of *universally* updating the entire network, we *selectively* update neurons based on their specific knowledge content. To implement our method, we require a heuristic to quantify the knowledge stored within each neuron. For this purpose, we introduce Historical Features Accumulation (HFA), which aggregates the gradient of the outputs and the neuron activations from different layers over multiple training steps. After each training step, we extract the aforementioned features from every neurons in selected layers, then perform a layer standardization and sum the normalized features across the training steps, as follows:

$$H\hat{A}_n = \sum_{t=1}^{T} \frac{A_n^l - \mu_A^l}{\sigma_A^l}, \quad H\hat{G}_n = \sum_{t=1}^{T} \frac{G_n^l - \mu_G^l}{\sigma_G^l}$$

where μ_A^l and σ_A^l are the mean and standard deviation of activations in layer l, and μ_G^l and σ_G^l are the mean and standard deviation of gradients in layer l. Using such heuristic as foundation, we next present the two building blocks of our framework: *Inner State Awareness* and *Targeted Updates with Differentiated Plasticity*.

Inner State Awareness

We propose Inner State Awareness as a mechanism for LLMs to classify new information as *novel*, *familiar*, or *conflicting* with respect to existing knowledge. Extracting the activations and gradients related to a specific fact, and exploiting HFA, we generate aggregated features which can lead to a precise classification of knowledge. Specifically, given a fact x, we aim at classifying it using a classifier $\mathcal{C}$

as following:

$$\mathcal{C} : (x, \hat{A}, \hat{G}, H\hat{A}, H\hat{G}) \rightarrow \{\text{novel}, \text{familiar}, \text{conflicting}\}$$

Considering that this classifier should work in online environments, we employ shallow machine learning classifiers such as Support Vector Machine (SVM) and Random Forest (RF) to provide fast responses.

Targeted Updates with Differentiated Plasticity
Differentiated plasticity is a method for tracking the historical usage of neurons to identify "free" and "busy" neurons, exploiting the gradient accumulation computed in HFA. The rationale behind this method is that when a neuron accumulates gradient, it indicates that its parameters are consistently moving in a specific direction, progressively integrating new adjustments. Differentiated Plasticity enables targeted updates to specific neuron groups, offering tailored advantages based on the type of learning task:

- **Free neurons**: the group of neurons that accumulate the least gradient during training. It should be leveraged for incrementally learn new knowledge, allowing new information to be incorporated while minimizing the risk of catastrophic forgetting.
- **Busy neurons**: the group of neurons that accumulate the most gradient during training. It should be utilized for model editing, allowing for precise targeting of knowledge that needs modification.

Experiments and results

For verifying the effectiveness of our method, we present three different sets of controlled experiments. Each experiment covers a specific part of the proposed framework:

- **Familiarity and dissonance detection**: this experiment aims at verifying the capabilities of our method in detecting coherent, dissonant and unfamiliar knowledge. After building a dataset composed of the three classes of facts, we train the model on the coherent facts. We then train a classifier to detect the different classes of facts using the proposed internal states analysis. Our method reaches 99.8% accuracy in detection proving to be a valid method for knowledge classification.
- **Incremental update**: in this experiment we compare targeted update to traditional fine-tuning for incremental learning. Initially, the model is trained on a baseline set of facts, representing the knowledge that must be retained.

Next, new facts are incrementally introduced. Across test cases, targeted updates consistently outperform full fine-tuning. Specifically, updating only the free neurons yields the best results in preserving prior knowledge, achieving an accuracy of 98% on the initial set of facts, which is significantly higher than the 60% accuracy measured after full fine-tuning.

- **Model editing**: it measures the effectiveness in modifying prior knowledge (reliability) without compromising unrelated information (locality), while ensuring the generalization of newly learned concepts (generalization). After training the model on a foundational set of knowledge to be preserved, we introduce a new set of facts and proceed to edit them, comparing the performance of targeted update with respect to both full fine-tuning and ROME [3], the current state-of-the-art method. Our results show that modifying the busy neurons achieves the highest performance, with a reliability of over 99%, comparable to full fine-tuning but with enhanced locality preservation. Notably, our approach significantly outperforms ROME, particularly when scaling to larger sets of facts. However, all the methods still struggle with generalization.

Conclusion

In conclusion, our work proved that (i) Inner State Awareness is a valid method for detecting the familiarity and dissonance of factual knowledge inside LLMs; (ii) Differentiated Plasticity for Targeted Updates is effective in incremental learning and editing for distinguishing between neurons that have learned previous knowledge and those that are still able to incorporate new facts. The application of the presented pipeline can significantly improve the incremental learning and editing of LLMs compared to current implementations. Despite the promising results, the proposed approach still faces some limitations. In particular, our method was tested on small architectures with low computational demands, and future work should focus on scaling it to larger models to evaluate its effectiveness in more complex scenarios. Another key research direction involves understanding the influence of the computational graph on model behavior, with the goal of minimizing internal interference by accounting for signal propagation. Additionally, current benchmarks are inadequate for assessing language models' true knowledge capabilities, further studies should explore novel generative evaluation methods that prioritize coherence and alignment over memorization of fixed fact structures.

Acknowledgements

Alla mia famiglia che mi ha supportato e sopportato durante questo viaggio: a mamma che c'è sempre per me e a papà che, anche se non lo ammetto mai, è un po' il mio supereroe. So che molto spesso non è facile avere a che fare con me, ma voi non mancate mai nel dare il vostro fondamentale apporto.

A Chiara, che è ormai diventata una componente imprescindibile delle mie giornate. In quest'anno abbiamo imparato che stare a distanza non è facile, ma come hai detto tu "preferirei stare tre giorni con te che tutti i giorni con un'altra persona". Mi sento poi in dovere di sottolineare, visto che so che ci tieni, che buona parte di questa tesi è stata scritta nell'isola più bella del mondo.

Ai miei amici che sono sempre stati presenti quando tornavo anche solo per un weekend. Menzione speciale a Davi: sei stato il mio solito compagno di avventure anche a distanza e sei una delle poche persone su cui so di poter contare sempre.

Un ringraziamento ai miei amici parigini Alfredo, Alberto e Matteo per avermi fatto sentire a casa lontano da casa. In particolare devo molto ad Alfredo, che con i suoi preziosi consigli e i suoi *vabbuò* mi ha guidato e accompagnato in questo strano mondo della ricerca. Sei una persona innamorata di quello che fa e sono sicuro che sarai un ottimo professore.

Vorrei infine ringraziare il Prof. Marco Mellia per aver reso possibile questo progetto e per la sua continua disponibilità; il Prof. Johan Boye per avermi accolto a braccia aperte, seguendomi con passione e dedizione; e per ultimo, ma sicuramente non per importanza, Zied, per il suo essere visionario e per il suo costante e incondizionato entusiasmo.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Human brain learning process

Humans effortlessly update their knowledge as they experience the world. They seamlessly integrate new information, ignore familiar facts, and actively engage in resolving conflicts with existing beliefs before updating their mental models. This cognitive flexibility stems from several key abilities. Humans exhibit (i) selective attention, focusing on novel or relevant information while filtering out redundant or irrelevant stimuli [4, 5, 6, 7, 8, 9, 10]. They readily (ii) identify inconsistencies between new information and existing knowledge through conflict detection [11, 12, 13]. Humans perform (iii) targeted updates, modifying specific neural pathways related to new information without disrupting unrelated knowledge [14, 15]. Lastly, (iv) their brains maintain adaptive plasticity, favouring stability of well-established knowledge, and showing more flexibility in case of uncertainty [14]. These capabilities allow human beings to manage a large knowledge base which is updated in a continuous manner through their entire life cycle. Important concepts learnt during our childhood are frozen in neurons which lose their plasticity protecting those fundamental concepts.

1.2 LLMs learning abilities

Despite demonstrating remarkable capabilities across various tasks, large language models (LLMs) are still far from such seamless learning abilities. Current LLMs face significant challenges in real-world deployment and long-term utility due to their static nature and training paradigms. They suffer from catastrophic forgetting, where incorporating new information often leads to the erasure of previously learned knowledge. Furthermore, LLMs engage in indiscriminate learning, passively

accepting all training data, even when it contradicts existing information. Despite emergent sparsity, knowledge in LLMs is diffusely distributed due to the nature of backpropagation, where all weights are potential candidates for storing knowledge, necessitating comprehensive retraining to incorporate new information. Unlike the human brain's varied adaptability, LLMs exhibit uniform plasticity, treating all parameters with equal importance during updates. Spreading the knowledge across all the network is not space efficient and it is not enforcing any kind of specificity for the neurons.

Furthermore, state-of-the-art models exploits tens, hundreds or even thousands billions of parameters. Every new model tends to be bigger than the previous: while this characteristic is generally linked to better performances [16], it increases drastically the computational need and the cost, making the development of such models exclusive privilege of large tech companies. At the same time, recent works proved that most of the parameters in big models are not strictly necessary for keeping performance unchanged [17]. The fact that a subset of parameters seems to be accountable alone for the output generation suggests that sparse update of LLMs could have a relevant impact on the models' behaviour.

1.3 Contribution

In this work, we embark on a systematic exploration of integrating the human inspired learning traits above into LLMs. We term this approach Cognitive Aware Incremental Knowledge Update. Our contribution lies in carefully implementing and assessing these traits using current LLM technology, leveraging backpropagation and introducing minimal disruptive changes to current architectures. Through a series of principled experiments, we investigate the potential benefits and challenges of emulating human learning mechanisms in artificial neural networks. In particular, we augment LLMs with two main contributions: (1) Inner State Awareness: a mechanism for LLMs to classify new information as novel, familiar, or conflicting with existing knowledge; (2) targeted update using Differentiated Plasticity: we develop a method for tracking the historical usage of neurons to identify "free" (rarely used) and "busy" (containing previous knowledge) neurons.

Our approach naturally bridges two important areas of research: techniques for editing knowledge within LLMs and methods for continual learning that preserve prior knowledge. It offers a native alternative to recent language model editing techniques such as ROME [3], MEND [18], and MEMIT [19]. While these methods focus on targeted edits to LLMs, they don't fully address the broader challenges

of continual learning and cognitive flexibility. Furthermore, most of the literature on this kind of methods focuses on editing single facts inside the models, making them not usable in real world scenarios. At the same time, we note that much of the existing work on continual learning has primarily focused on traditional neural networks. For what concerns LLMs instead most of the current studies keep their focus on the layers, without diving deeper into finer granularity such as the neurons level. To evaluate the efficacy and limitations of our approach, we conduct a series of carefully designed experiments. We assess the ability of LLMs to identify and classify new information correctly through familiarity and dissonance detection. We examine how effectively LLMs can update knowledge incrementally without forgetting previous information. We investigate how well LLMs handle conflicting information and update beliefs accordingly.

Through our experiments, we uncover several important findings. Sparsity in updates can help pack new knowledge while preserving prior knowledge, sometimes even when applied randomly. Targeting "free" neurons for incremental updates shows potential in preserving prior knowledge, while indiscriminate updating of "busy" neurons can be more destructive. On the other side, regions of the network that were heavily trained during pre-training appear more effective at incorporating knowledge updates, echoing findings from the lottery ticket hypothesis. Updating conflicting information (dissonant) poses greater risks to prior knowledge, highlighting the importance of conflict-aware learning strategies. Our current implementation of incremental editing shows improvements over existing methods like ROME in specific scenarios. In addition to promising results and insights, we also uncovered significant challenges. Updates to "free" neurons can sometimes interfere with "busy" neurons, altering their behavior in unexpected ways. This suggests that the separation between knowledge storage areas may not be as clear-cut as initially hypothesized. The complexity of large neural networks makes it difficult to predict the full impact of targeted updates, highlighting the need for more sophisticated monitoring and control mechanisms. Finally, while our approach shows promise in controlled settings, scaling it to full-scale LLMs and diverse real-world scenarios remains a significant challenge.

This work represents a step towards bridging the gap between human-like learning and current LLM capabilities. By implementing and evaluating human-inspired learning mechanisms within existing LLM frameworks, we provide a practical playground to experiment with new human-inspired incremental learning approaches. Our findings open up new avenues for research in continual learning, knowledge representation, and cognitive science-inspired AI.

Chapter 2

Background

In this chapter we delve into the details of the existing literature, providing a comprehensive overview of the background knowledge related to our work. We explore the fundamental concepts, key studies, and recent advancements that form the basis of our research, including the general concepts related to LLMs and detailed description of current incremental update and editing techniques.

2.1 Large Language Models

LLMs are models which have demonstrated exceptional capabilities in learning complex language patterns thanks to their high number of parameters. To explain how they operate more clearly, we can describe an LLM as a function F that transforms an input sequence of tokens $x = (x_1, x_2, ..., x_n)$ into an output sequence of tokens $y = (y_1, y_2, ..., y_m)$, using a specified vocabulary V: this vocabulary represents the set of tokens which the model has at its disposal to generate the output. Tokens are generally composed by entire words or part of them, plus some special tokens such as the End-Of-Sentence (EOS).

Initially, each input token x_i is converted into a vector in a high-dimensional space via an embedding layer, creating a series of vector embeddings E. Generally LLMs are based on the transformer architecture: they are built by stacking several transformer blocks each one consisting of a multi-head self-attention module A and a multilayer perceptron (MLP) M. In each layer l, the transformation begins with the previous layer's hidden state H_{l-1} (with $H_0 = E$) undergoing processing by the self-attention mechanism to produce A_l. This output is then further processed by the MLP layer. The MLP consists of several fully connected layers that refine this data, resulting in M_l. The entire computations made by a single layer can be summarized as:

$$\begin{cases} A_l = A(H_{l-1}) \\ M_l = M(A_l,\ H_{l-1}) \\ H_l = H_{l-1} + A_l + M_l \end{cases} \tag{2.1}$$

After the transformations applied by the last layer L the output passes through a final linear layer with output dimensionality equal to the vocabulary size V: the resulting vector represents the probability distribution of the next token over the entire vocabulary V.

2.2 Backpropagation

Neural network training process is based on the concept of backpropagation, a gradient estimation method used by the optimization algorithms to compute the network parameter updates. The final goal is to minimize the Loss L by modifying the networks parameters: this can be done with an update in the opposite direction of the gradient of L with respect to weights and biases. While the use of such technique is common knowledge, it is worth analyzing in detail the process to explain clearly each step.

As starting point we need to define the loss L which in this case will be indicated by a general cost function $C(y, a_N)$ where y represent the labels and a_N the activations of the last layer (output of the network). The gradients under analysis are then defined as ∇_w and ∇_b, which represent respectively

$$\nabla_w = \frac{\partial L}{\partial w} \quad \nabla_b = \frac{\partial L}{\partial b} \tag{2.2}$$

and they can be obtained through an iterative backward process from the last layer to the first one. First of all, after defining the activation function as a general σ, we can start by computing the output a_L through the equation

$$a_N = \sigma(w_N a_{N-1} + b_N) = \sigma(z_N) \tag{2.3}$$

Once obtained the activations of the last layer we can start computing the first gradients with respect to weights and biases using the chain rule:

$$\nabla_{w_L} = \frac{\partial L}{\partial w_N} = \frac{\partial L}{\partial a_N} \cdot \frac{\partial a_N}{\partial z_N} \cdot \frac{\partial z_N}{\partial w_N} \tag{2.4}$$

$$\nabla_{b_L} = \frac{\partial L}{\partial b_N} = \frac{\partial L}{\partial a_N} \cdot \frac{\partial a_N}{\partial z_N} \cdot \frac{\partial z_N}{\partial b_N} \tag{2.5}$$

Exploiting this mathematical property, we are able to extract the the target gradient as function of other gradients. Both equations can be simplified by doing the intermediate computations:

$$\nabla_{w_N} = C(y, a_N)' \cdot \sigma'(z_N) \cdot a_{N-1} \tag{2.6}$$
$$\nabla_{b_N} = C(y, a_N)' \cdot \sigma'(z_N) \cdot (1) \tag{2.7}$$

Now that we have the result for the last layer, we only need to iterate the passage to the previous layers in the exact same way. Despite following the same rule, equations will get more complicated while propagating since there is a chain dependence in the computation of the activations. Here is an example of the penultimate layer:

$$\nabla_{w_{N-1}} = C(y, \sigma(w_{N-1}a_{N-2} + b_{N-1})' \cdot \sigma'(z_{N-1}) \cdot a_{N-2} \tag{2.8}$$
$$\nabla_{b_{N-1}} = C(y, \sigma(w_{N-1}a_{N-2} + b_{N-1})' \cdot \sigma'(z_{N-1}) \tag{2.9}$$

After completing propagation it is possible to obtain the gradient with respect to all the trainable parameters: this computation retrieves the direction for the update which is then modulated by the choice of the learning rate.

2.3 Transformer

Since it was first introduced [20], transformer architecture has become one of the most effective and common frameworks in the field of NLP. It was originally built to avoid using recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The architecture is based on an encoder-decoder structure shown in Figure 2.1. The encoder processes the input sequence, transforming it into a series of continuous representations. These representations are then provided to the decoder, which integrates them with its own outputs from preceding steps to generate the final sequence.

The encoder consists of two sub-layers: the multi-head self-attention mechanism and the FFN layer. The first sub-layer takes is fed with the input embedding enriched by the positional encoding and it is responsible for the attention mechanism that assigns weights to each word in a sentence based on its importance. It first generates three vectors which are abstract representations used in the computation: queries, keys and values. The output is calculated as a weighted sum of the values, with each value's weight determined by a compatibility function that measures the query's alignment with the corresponding key.

$$Z = softmax\,(\frac{Q\,K^T}{\sqrt{d_k}})\,V \tag{2.10}$$

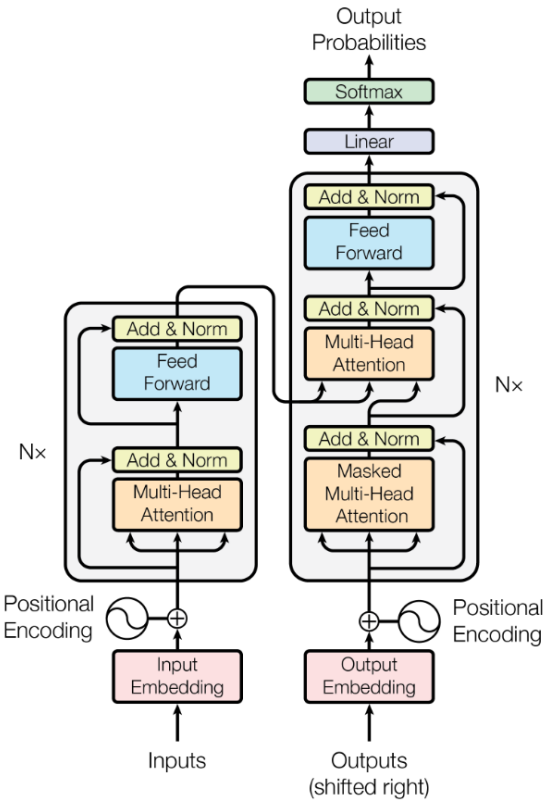

Figure 2.1: Transformer architecture: encoder on the left, decoder on the right. Source [20]

This procedure is executed simultaneously by utilizing various heads to create multiple representational subspaces. The results from each head are concatenated, and through the application of a linear layer, a projection back to the original dimension is achieved. Each attention head will produce its own result, which will be concatenated with the others; an additional weight matrix is used to reduce the dimensionality through a projection to obtain the final representation (Figure 2.2.

$$Z = concat\left(Z_0, Z_1..., Z_n\right) W_{proj} \tag{2.11}$$

The second sub-layer instead 2.1, is a fully connected feed-forward network consisting of two linear transformations with ReLU activation in between:

$$FFN(Z) = ReLU\left(W_2\, Z + b_2\right) \tag{2.12}$$

Furthermore, each sub-layer has a residual connection summing up the input value with the output and it is followed by a normalization layer which normalizes the values of the sum matrix.

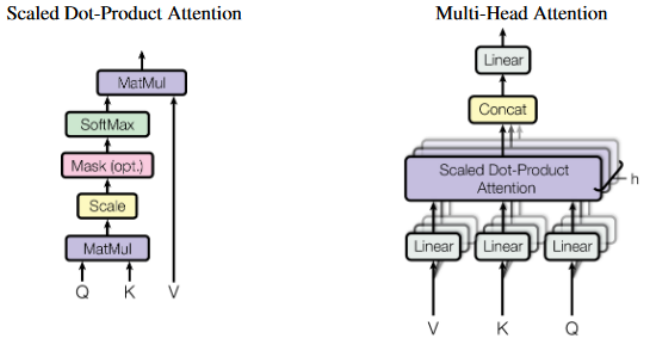

Figure 2.2: Self-attention block. Source [20]

The decoder shares many similarities with the encoder: it is characterized by a composition of several identical layers stacked upon one another, each utilizing an attention mechanism. However, the internal structure of the blocks differs since both the encoder outputs and the decoder previous outputs must be taken into account at this stage: for this reason we will have three sub-blocks. The first sub-layer receives the previous output of the decoder stack, augments it with positional information, and implements multi-head self-attention over it. While the encoder is structured to consider every word in the input sequence, regardless their positions and allowing a comprehensive understanding of the entire sequence, the decoder is modified to attend only to the preceding words. This means that the prediction of a word at any given spot is influenced exclusively by the words that appear before it, ensuring that the generation process is sequential and each new word prediction builds on the known context. The masking operation is implemented by suppressing the values that would correspond to connections that are not allowed.

$$mask(Q\,K^T) = \begin{bmatrix} a_{11} & -\infty & -\infty & \cdots & -\infty \\ a_{21} & a_{22} & -\infty & \cdots & -\infty \\ a_{31} & a_{32} & a_{33} & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \tag{2.13}$$

The second layer implements a multi-head self-attention mechanism similar to the one implemented in the encoder. This time, the attention mechanism receives queries from the previous decoder sub-layer, while keys and values are taken from the output of the encoder. This allows the decoder to attend to all the words in the input sequence. Finally, the third layer is a FFN layer with the same structure of the one in the encoder. Residual connections and normalization layers are present for each sub-layer following the same logic of the encoder.

2.4 GPT architecture

The Generative Pre-trained Transformer (GPT) model represents one of the most wide-spread architectures in LLMs. GPT is an autoregressive decoder-only architecture based on a stack of transformer-decoder blocks which can vary in number and dimensionality depending on the specific model. The choice behind the deletion of the encoder model relies on the fact that this kind of architecture is mainly used for text generation: the decoder blocks allow the model to sequentially predict the next token in a sentence given all the preceding tokens, without the need of additional elements which impact on the model efficiency.

The first GPT model [21] was built upon the intuition that language models could be used to solve different kinds of tasks. The core innovation was the development of a general pre-training on a diverse corpus of unlabeled text to create a knowledge base, followed by discriminative fine-tuning for a specific task. This approach allows the model to adapt to different kinds of tasks with minor internal changes thanks to transfer learning. The original GPT block structure follows the transformer-decoder block with the elimination of the self-attention block which has no reason of being used given the absence of the encoder (Figure 2.3).

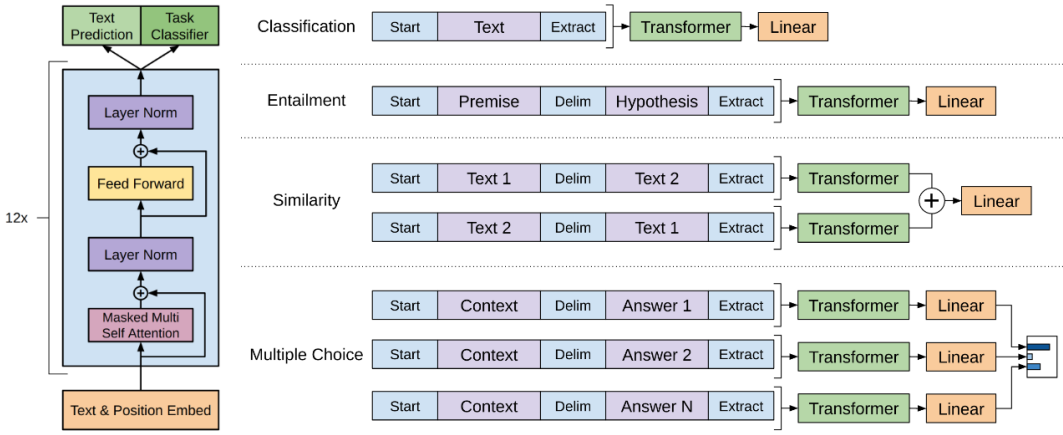

Figure 2.3: GPT architecture. Source [21]

The second iteration of this kind of architecture is represented by GPT2, a family of models of different size which improves the original GPT architecture (Figure 2.4). The main difference from a conceptual point of view is the change of objective of the training: GPT2 aims at utilizing unsupervised pre-training for supervised tasks. According to this new paradigm a language model could do multi-task learning directly from the training process, enabling general capabilities

without a fine-tuning. In this evolution the layer normalization has been moved to the input section of each block, and an additional layer normalization is added after each self-attention.

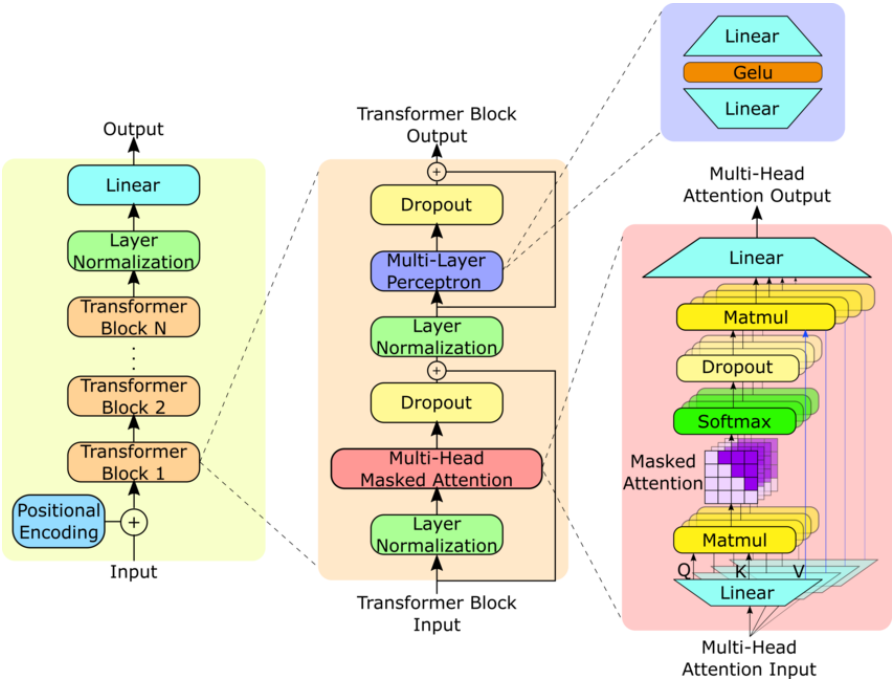

Figure 2.4: GPT2 architecture. Source [22]

The most recent updates, represented by GPT3 and GPT4, have not been made public from an architectural point of view. However, thay can be tested in inference and GPT4 represents the current state-of-the-art in different benchmarks. In this work, we are going to analyze the architecture of GPTJ, a model released by the open source community as an alternative to GPT3.

2.5 Language modelling

LLMs have different architectural choices which works using different working principles. This diversity is not just a matter of scale or efficiency, but it reflects the strategy these models employ to mimic human linguistic abilities. This aspect is commonly defined as language modeling and in this chapter we present the main approaches used in current models (Figure 2.5).

Causal Language Modelling. Causal Language Modeling (CLM) centers on

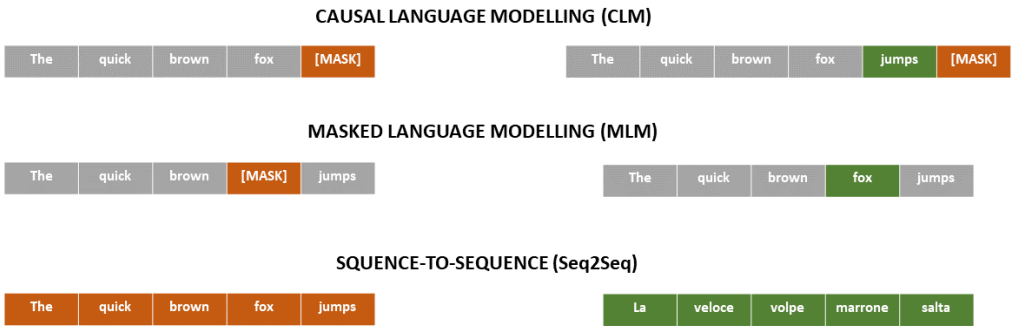

Figure 2.5: Graphic representation of the different types of language modelling.

predicting the next word in a sequence based on the preceding words. This is a one-directional method, where the model learns to generate text by focusing solely on the words that come before the one it is trying to predict. This sequential nature compels the model to understand not just the probability of word combinations but also how ideas and narratives unfold over time. CLM is typically employed in autoregressive models like GPT, where each token can only reference the tokens that appear earlier in the sequence. This structure prevents the model from "cheating" by looking at future tokens during training, as it relies entirely on past context. These models excel at generating coherent, contextually appropriate text, making them well-suited for tasks such as text generation, storytelling, and even code completion.

Masked Language Modelling. In Masked Language Modeling (MLM), certain words in a sentence are randomly hidden during training, and the model's task is to predict these masked words using the context provided by the visible words in the sentence. This technique allows the model to develop a strong understanding of language context and structure by forcing it to deduce the missing information from the surrounding words. These abilities are particularly useful for tasks like text classification and sentiment analysis, since MLM models can leverage the full sentence context, allowing them to capture information from both preceding and following words.

Sequence-to-Sequence. Seq2Seq models consist of two main parts: an encoder that processes the input sequence and a decoder that generates the output sequence. This design enables the model to manage input and output sequences of varying lengths, making it highly adaptable to many tasks beyond just language processing. This approach excels in tasks that require transforming one sequence into another, such as machine translation, text summarization, and speech recognition. With the integration of attention mechanisms, these models have become even more effective

at handling longer sequences. Seq2Seq models are well-suited for more complex tasks involving input-output transformations, making them highly versatile across a broad spectrum of natural language processing tasks.

2.6 Lottery ticket hypothesis

LLMs are getting bigger making the training less sustainable and more expensive: for this reason, some techniques such as pruning, have been created to mitigate this problem. Neural network pruning is a technique used to reduce the size of trained neural networks by removing unnecessary parameters, which helps in decreasing computational load and storage requirements while maintaining accuracy. However, the pruned networks are often difficult to train from scratch and they still require the training of a full model. This condition poses important limitations, since the advantage in terms of computational capacity are just exploitable at inference time. The training, which is the part requiring most resources, is not improved.

The "Lottery Ticket Hypothesis" proposed by Frankle and Carbin [23], tries to solve this issue by suggesting that dense, randomly-initialized neural networks contain smaller subnetworks that are capable of training effectively from their initial states. These subnetworks, termed "winning tickets," are identified through a process of iterative pruning, where a portion of the network's weights is pruned based on their magnitude, and the remaining weights are reset to their original values from initialization. These winning tickets, once trained in isolation, can match or even exceed the accuracy of the original network within a comparable number of training iterations. Experiments supporting the hypothesis demonstrated that winning tickets in various architectures, including fully connected and convolutional networks, can constitute as little as 10-20% of the original network's size. Furthermore, the study finds that the success of these winning tickets is heavily dependent on their initial weights; when reinitialized randomly, the subnetworks lose their advantageous properties, emphasizing the importance of the initial state.

The implications of the Lottery Ticket Hypothesis are significant for the design and training of neural networks. It suggests that the training of smaller, more efficient networks could be possible if these winning tickets can be reliably identified early in the training process. This hypothesis not only offers a potential path to more resource-efficient neural networks but also provides insights into the structure and initialization that contribute to the effectiveness of the training.

2.7 Incremental learning

Incremental learning refers to a system's capacity to continuously acquire knowledge over time, allowing it to adapt and generalize to new tasks [1]. This enables the system to adapt and react to changing conditions and user behavior in real time, without the need for periodic complete retraining sessions. In particular, in this work we analyze the continual learning capabilities of LLMs, namely the capacity of learning new, emerging tasks efficiently while reducing the catastrophic forgetting. Key objectives of continual learning include:

- *Minimizing Catastrophic Forgetting*: This is a major challenge in neural networks, where learning new tasks can lead to the loss of previously acquired knowledge. Catastrophic forgetting occurs when updates for new tasks interfere with weights important for old tasks, resulting in performance degradation.

- *Efficient Knowledge Integration*: Continual learning systems should integrate new information without the need to retrain on the entire dataset, which can be computationally expensive and time-consuming.

- *Scalability and Memory Efficiency*: As the number of tasks grows, CL methods must maintain scalability and memory efficiency, avoiding excessive growth in model size or memory usage.

Continual learning methods can be broadly classified into three categories: regularization-based, dynamic-architecture-based, and memory-based methods. Each of these approaches tackles the challenge of integrating new information while preserving existing knowledge in different ways:

Regularization-based Methods: These methods protect critical parameters of the neural network by adding a regularization term to the loss function. They penalize changes in parameters crucial for maintaining the performance of old tasks. Techniques such as Elastic Weight Consolidation (EWC) [24] and Knowledge Distillation [25] are widely used for this purpose. Regularization-based approaches are effective in preventing catastrophic forgetting but may struggle when the number of tasks grows significantly.

Dynamic-Architecture-based Methods: These methods expand the network architecture to accommodate new tasks, adding task-specific modules or parameters without altering existing ones. For instance, methods like Progressive Neural Networks introduce new columns for each task, and Mixture-of-Experts (MOE) [26] dynamically allocates expert modules based on task requirements. While these methods can effectively mitigate forgetting, they lead to increased model complexity

and computational costs.

Memory-based Methods: Memory-based methods retain a subset of data from previous tasks to replay during training on new tasks. This can be achieved through experience replay, where old examples are interleaved with new ones, or by using generative models to create synthetic samples of previous data. Memory-based methods are powerful but depend heavily on the quality and representativeness of the stored data.

Finally, when discussing specifically about continual learning for LLMs it is appropriate to consider unique challenges due to their scale and the computational resources required for fine-tuning.

2.8 Model editing

Model editing [2] can be described as the task of modifying the behavior of an initial base model in response to a particular edit descriptor (x_e, y_e), while ensuring that the performance on other data points remains unaffected. The objective is to transform the original model f_θ (where θ represents the model's parameters) into an updated version f_{θ_e} that aligns with the specified descriptor. In the context of a large language model (LLM), this can be viewed as a function that maps a given input to a probability distribution for the output. The base model generates a prediction y using $f_\theta(x)$, whereas the updated model produces a modified result $y_e = f_{\theta_e}(x)$, guided by the descriptor x_e. An illustration of this concept is shown in Figure 2.6.

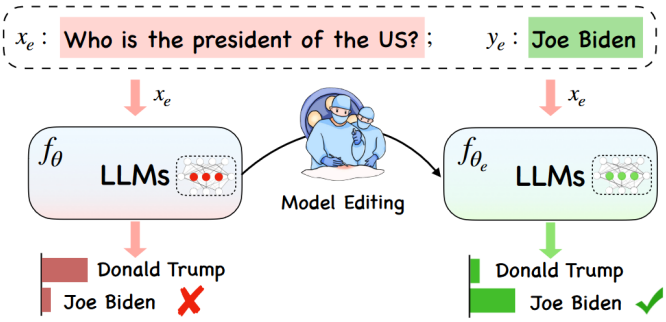

Figure 2.6: Visual representation of model editing on LLMs. Source [2]

The process of editing a model typically influences the predictions for a wide range of inputs that are closely linked to the example used for the edit. This group of inputs is referred to as the **editing scope**. An effective edit aims to

modify the model's responses for inputs within the editing scope (we do not want just to modify a single fact but more in general, the knowledge related to that fact), without affecting its performance on examples that fall outside this scope.

$$f_{\theta_e}(x) = \begin{cases} y_e & \text{if } x \in I(x_e, y_e) \\ f_\theta(x) & \text{if } x \in O(x_e, y_e) \end{cases} \tag{2.14}$$

$I(x_e, y_e))$ represents the group of facts connected to the edited one and it is defined as *in-scope*. Unrelated facts which should not be affected by the modifications instead, are defined as *out-scope* and they are represented by $O(x_e, y_e))$.

Some metrics have been created in order to measure the efficacy of the editing process: **reliability**, **generalization**, and **specificity**.

Reliability Earlier studies consider an edit reliable if the modified model f_{θ_e} produces the correct answer for the specified edit instance (x_e, y_e). This concept of reliability is quantified by calculating the mean accuracy for the edited case.

$$\mathbb{E}_{\ x'_e, y'_e \sim \{(x_e, y_e)\}} \ \mathbf{1}\{\text{argmax}_y \ f_{\theta_e}(y \mid x'_e) = y'_e\} \tag{2.15}$$

Generalization The edited model f_{θ_e} should not focus only on the given fact, but also modify the equivalent pairs from the neighbour $N(x_e, y_e)$. Its performance is measured by the mean accuracy across samples uniformly selected from the equivalence class.

$$\mathbb{E}_{\ x'_e, y'_e \sim N(x_e, y_e)} \ \mathbf{1}\{\text{argmax}_y \ f_{\theta_e}(y \mid x'_e) = y'_e\} \tag{2.16}$$

Specificity Editing needs to occur on a local level, implying the post-edit model f_{θ_e} should avoid altering the outcomes for examples outside the relevant scope, $O(x_e, y_e))$. Therefore, the model's specificity is assessed based on the frequency at which the predictions by the post-edit model remain consistent with those of the pre-edit model.

$$\mathbb{E}_{\ x'_e, y'_e \sim O(x_e, y_e)} \ \mathbf{1}\{\ f_{\theta_e}(y \mid x'_e) = f_\theta(y \mid x'_e)\} \tag{2.17}$$

Currently there are already several methods for performing knowledge editing on LLMs. Yao *et al* [2] present a deep analysis of the different techniques available offering a strong baseline for our analysis. In the mentioned work methods are divided into two main families (Figure 2.7): approaches which modify the model's parameters and techniques preserving the parameters.

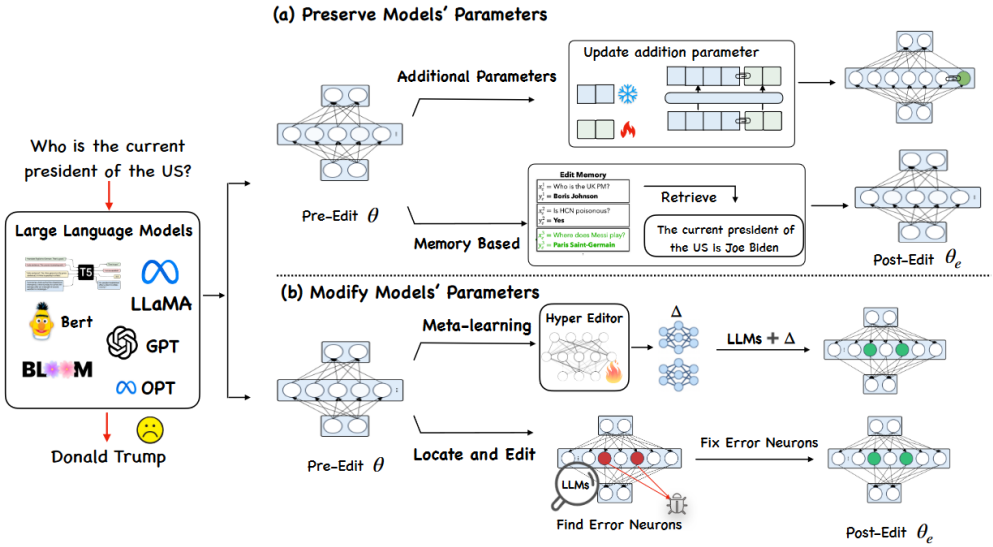

Figure 2.7: Visualization of the different editing paradigms. Source [2]

2.8.1 Memory-based methods

This approach retains every instance data to be edited in memory, using a retrieval mechanism to retrieve the most pertinent edit information for each incoming input, directing the model to produce the modified fact when required. In order to reach this result it is possible to adopt different techniques.

In-context learning

The most straightforward method of implementing memory-based editing is by leveraging the strong *in-context learning* capabilities of LLMs. Using a traditional Retrieval Augmented Generation approach, the model can generate responses based on the provided knowledge by incorporating a refined knowledge context directly into the prompt. This approach does not involve any changes to the model's architecture or the addition of extra models, but instead alters the output by prompting the model with the updated information. This technique has been demonstrated in systems such as MemPrompt [27] and MQuAKE [28].

SERAC

Considering a more structured approach, Mitchell *et al* presented SERAC [29] which exploits two different models and a cached memory containing edited facts: the first model is identical to the original, while the other is trained specifically on the

edited facts list. This approach uses then a *scope classifier* to assess the likelihood that new input is covered by examples of edits it has stored. If the new input does not align with any stored edits, the response is generated using the system's original prediction method. On the contrary, if the new input is similar to any of the stored edits, the system's response is generated based on the modified model: in this way it is possible to maintain the exact original behaviour for the non-edited facts, while being able to answer correctly to specific modified facts when asked. The reason for adopting the SERAC approach is rooted in the observation that neural networks have the ability to 'over-specialize' their parameters for specific inputs, leading to the possibility of different, non-overlapping sections of the model being in charge of predictions for a given input. Consequently, gradients might not offer adequately comprehensive information to facilitate dependable edit scoping, especially for examples that are related but not closely aligned.

2.8.2 Additional Parameters

This paradigm implies an internal change in the model architecture, with the insertion of extra parameters inside the LLM which can be trained for learning new knowledge: this is done without modifying the weights of the original model. We can consider this approach as an hybrid solution, where parametric knowledge is exploited without impacting on the previous configuration.

Transformer-Patcher

Huang *et al* [30] proposed the Transformer-Patcher methodology which implies the insertion of a new neuron the last FFN layer for each wrongly generated token related to a fact: this neuron called *patch* is trained in order to be active only in presence a given *mistake*. Each patch is trained considering reliability, generality and locality principles and has the role to modify the output to match the required edited fact. All the process is done freezing the original weights, which maintain their original values, and it can support the training of several patches at the same time. This approach is based on the assumption that FFN layer works like a memory system of key-value pairs at the neuron level. The computation it performs during the forward pass involves retrieving values from the matrix V by finding matches between the keys in matrix K and the input query q. The standard behaviour follows this equation:

$$a = Act(q \cdot K + b_k) \tag{2.18}$$

$$FFN(q) = a \cdot V + b_v \tag{2.19}$$

Once the patch is applied instead we have the insertion of the new terms k_p, v_p and b_p which represent respectively the key patch, the value patch and the bias patch. The final effect an be reformulated as a simple correction term added to the default FFN computation.

$$\begin{bmatrix} a & a_p \end{bmatrix} = Act(q \cdot K + \begin{bmatrix} b_k & b_p \end{bmatrix}) \tag{2.20}$$

$$FFN_p(q) = \begin{bmatrix} a & a_p \end{bmatrix} \cdot \begin{bmatrix} V \\ v_p \end{bmatrix} + b_v \tag{2.21}$$

$$FFN_p(q) = FFN(q) + a_p \cdot V_p \tag{2.22}$$

CaliNET

CaliNET, by Dong *et al* [31], starts from the same assumption related to key-values memory capabilities of FFN. This time there is the addition of a novel knowledge verification technique called Contrastive Knowledge Assessment: this process aims at finding errors in the facts storing of LLMs. Once wrongly learnt facts are detected, they are corrected through a calibration process which implements a new architecture which is similar to the one implemented by the original FFN but smaller. The idea is to tune this additional smaller network to adjust the output and correct the factual knowledge error.

2.8.3 Constrained fine-tuning

Fine-tuning has traditionally been a key method for adapting large language models (LLMs) to specific domains or integrating particular knowledge. To learn new facts, a basic approach involves fine-tuning the model on a targeted dataset D_M, which consists of the altered facts. This allows the model's predictions to be adjusted so that its output aligns with the desired behavior [32]. In this situation we can describe the optimization formula as:

$$min_{\theta \in \Theta} \frac{1}{m} \sum_{x \in D_M} L(x; \theta) \tag{2.23}$$

In this context, $m = |D_M|$ represents the number of supporting evidences related to the facts that need to be updated, θ refers to the model parameters being adjusted, and $L(x; \theta)$ signifies the per-instance loss used during fine-tuning. While this method can effectively help the model memorize new facts, it does not address the potential collateral damage to the unchanged set of facts, which can result in catastrophic forgetting. To enhance this aspect, a possible solution is to develop a fine-tuning set that includes both the modified D_M set and the unmodified facts $D_{F \setminus S}$, helping to ensure the model retains knowledge of unaltered information. However, research has shown that this approach alone is insufficient to fully prevent

forgetting, as the model often follows an imbalanced optimization path that favors the modified facts.

For this reason, Zhu *et al* [32] introduced constrained fine-tuning, a method designed to update specific factual knowledge within the model while preserving its performance on unaltered facts. This is accomplished by using a novel optimization function that incorporates both the modified and unmodified facts, enforced through an additional constraint to ensure balanced learning

$$min_{\theta \in \Theta} \frac{1}{m} \sum_{x \in D_M} L(x; \theta) \quad s.t. \ \frac{1}{n} \sum_{x' \in D_{F \setminus S}} L(x'; \theta) - L(x'; \theta_0) \leq \delta \tag{2.24}$$

In this equation δ is a small positive constant, which determines the tolerance for unmodified facts' variations. The core idea is to modify the model in the same way as a normal fine-tuning, taking care of not changing facts which should be kept as they are. Considering the computational cost of the loss, the function can be approximated to

$$min_{\theta \in \Theta} \frac{1}{m} \sum_{x \in D_M} L(x; \theta) \quad s.t. \ || \theta - \theta_0 || \leq \delta \tag{2.25}$$

where the constraint is put directly to model's parameters norm to improve performances (both l_2 and l_∞ have been tested, with the latter resulting in a more stable behaviour).

2.8.4 Locate-Then-Edit

In the classical fine-tuning, like in all global modification techniques all parameters (or at least those coming from the selected group or layers) are modified in the same manner: this means that the editing process involves the model at global scale. The Locate-Then-Edit approach instead, aims at improving the locality of the editing by finding which parts of the network are responsible for the knowledge of given facts.

ROME

Rank-One Model Editing has been presented by Meng *et al* [3] as a method to edit LLMs by altering the parameters that determine a layer's behavior at the decisive token. In order to locate the correct neurons to be edited the causal effects of hidden state activations is exploited: the target is to find which parts of the network recall a given fact when a specific subject is given as input. More precisely, the expected output of this method is to identify which combination of token and

layer so that the activations of the given token at the specified layer are the most important when generating the completion.

The process is divided in several runs. The first step is a clean run, when a prompt x is fed to the model, in order to obtain the expected completion o: this passage is crucial to collect the original model's activation. In the second run, which is defined as corrupted, the embeddings of all the tokens related to the subject are modified by the addition of a random noise. The original set $[h_1^{(0)}, h_2^{(0)}, h_3^{(0)}...]$ is modified so that each component becomes $h_i^{(0)} = h_i^{(0)} + \epsilon$, where $\epsilon = N(0, v)$. After breaking the information related to the subject, the model's completion is expected to be modified. The last passage consist in a run with partial restoration: starting from the broken embeddings, the activations $h_i^{(l)}$ related to a specific token i at a given layer l are restored. The goal is to evaluate how well a few intact states can restore the correct fact, even when many other states are corrupted by an obfuscated subject. The significance of each state is assessed using a metric known as the *indirect effect* [3]. This metric quantifies the difference in the probability of achieving the expected outcome in the partially restored state versus the corrupted state. It can be described as follows:

$$IE = P_{*,\, clean\ h_i^{(l)}}[o] - P_*[o] \tag{2.26}$$

where $*$ represents the corrupted state and o the expected completion. This process enabled the possibility to perform analysis on the importance of given parts of the model and the prompts. FF layers seems to be those with the major influence on factual associations. On the other side, the last token related to the subject has been identified as the most influential to completion generation.

Once the location part has been completed, it is time to actually update the model's knowledge. The idea is to consider the the transformer MLP as an associative memory represented by the parameters of the second MLP layer $W_{proj}^{(l)}$ so that given a key vector V we can obtain the value vector V as $V = WK$. The target is to insert a new key-value pair (k_*, v_*) into such memory. This action can be achieved by modifying W according to the equation

$$\hat{W} = W + \Lambda(C^{-1}k_*)^T \tag{2.27}$$

$$min||\hat{W}K - V||\ s.t.\ \hat{W}k_* = v_* \tag{2.28}$$

The target is to minimize the impact on the overall values extraction while inserting the new key-value pair. Before updating the actual weights, it is necessary to compute two additional variables. The value k_* represents the key, which encodes the subject: it is a value obtained by extracting the average activations related to

subject's last token at the chosen layer in different short prompts (to get a general result). v_* instead, is the target value we should generate as output of the selected MLP: it is built by maximizing the probability of obtaining o as final output, while minimizing the general effect on the other predictions.

2.8.5 Meta-learning

In meta-learning the task of knowledge editing inside pre-trained models is delegated to external networks: instead of perform the editing directly by handcrafted methods or to modify the model internal structure, it is possible to delegate the work to a network trained on the LLM weights to predict the best update.

Knowledge Editor

De Cao *et al* [33] introduce Knowledge Editor (KE), a technique that involves training a hyper-network (a network that predicts the parameters of another network) on the model's weights. This is done with constrained optimization to ensure that a specific fact can be updated without altering the remaining knowledge. Once the hyper-network is trained, it can be utilized during testing to predict the necessary weight adjustments. The constrained problem for editing a model θ to ensure that a specific input x produces the output $f(x, \theta) = a$ can be formulated as follows:

$$min \sum_{\hat{x} \in P^x} L(\theta'; \hat{x}, a) \quad s.t.\ C(\theta, \theta', f, O^x) \leq m \tag{2.29}$$

where θ' corresponds to the edited model, P^x is the set of paraphrases of x, O^x is the set of facts which should not be modified and C a constraint for the update which should be less or equal than a positive constant m that is chosen as hyperparameter of the method. Even if in the equation all paraphrases are considered, at training time the network is provided with the fact only since the same thing will happen during the test phase.

The network is not built to output θ' but if focuses on the prediction of the weights shift $\Delta\theta$ such that $\theta' = \theta + \Delta\theta$. To reduce the number of parameters, the network exploits the gradient $\nabla_\theta(\theta; x, a)$ using the assumption that it contains information related to the parameters to be updated for changing the likelihood of f for a. The model woks though a bidirectional LSTM which takes as input the encoded information related to input, current output and expected output $< x, y, a >$. The output of the last LSTM hidden state is fed to a FFNN which produces an output vector h; this vector conditions a second FFNN that is used to

extract the parameters used for weights and bias update computation.

MEND

In contrast, Mitchell *et al* [18] introduced MEND, a technique that learns to adjust the raw fine-tuning gradient, enabling a more precise parameter update that allows the model to be effectively edited in a single step. The strategy behind MEND exploits an external mechanism to refine the fine-tuning gradient for a specific layer: the objective is to turn it into a parameter update that ensures reliability, locality, generality, while keeping the process as efficient as possible. The input to the network is a high-dimensional matrix representing the layer's gradients, and the output is the resulting parameter update. Given that each type of layer has its own dimensionality, the method exploits a set of different networks, each one tailored on a given type of layer and compatible with its dimensionality. Considering the high-dimensionality required by the process it is necessary to improve the efficiency in some way: for this reason, instead of using the actual gradients, the network works with output gradients. This can be done since both the actual gradient matrix g_l and the gradient of the loss L with respect to the weight of a given layer W_l for each element of a given batch B are rank-1 matrices. The gradient can be then computed as

$$\nabla_{W_l} L = \sum_{i=1}^{B} \delta_{l+1}^{i} \, \mathbf{u}_l^{i\top} \tag{2.30}$$

where δ_{l+1}^{i} is the gradient of the loss with respect to the output of the level l (which is equal to the set of pre-activations of layer $l+1$) referred to a specific batch element i while u_l^i represents the input of the layer itself.
Each network is then trained using a constrained loss which aims at both maximizing the edit success and minimizing the effects on unrelated facts. The final aim is to learn how to modify the classical fine-tuning gradients in order to reduce the given loss: this approach requires a relevant amount of training time but allows to have refined gradients with few resources at inference time.

Chapter 3

Proposed approach

In this chapter, we provide a detailed explanation of the proposed approach. We begin by defining the general concept of cognitive-aware learning, which involves selectively updating the neural network based on the knowledge content of its neurons. Next, we delve into the specifics of inner state collection, outlining how we gather and process the relevant internal states of the network. After establishing these basics, we describe the working principle of our familiarity and dissonant detection method for Cognitive Awareness, which enables us to assess how a given fact relates to the model's existing knowledge. Finally, we present the implementation of the targeted network update through Differentiated Plasticity, which is built to effectively insert new knowledge into the model or edit prior beliefs.

3.1 Cognitive-aware learning

Our technique builds upon the concept of cognitively aware updates, where instead of universally applying backpropagation to update the entire network, we selectively update neurons based on the specific knowledge content they hold. By differentiating the neurons according to the relevance and importance of the information they hold, this method aims at making the learning process more efficient and targeted.

3.2 Inner states data collection

To implement our approach, we need a heuristic capable of estimating the knowledge content of each neuron. In this work we present the Historical Features Accumulation (HFA), built by accumulating the gradient outputs (grad_outs) and activations of each neuron inside the transformer blocks' layers, over the training steps. In this work, we focus on the historical tracking of gradients of the outputs

(grad_outs) and activations for four key matrices within each block of the transformer model: $\text{Attn}_{\text{c\_attn}}$, $\text{Attn}_{\text{c\_proj}}$, $\text{MLP}_{\text{c\_fc}}$, and $\text{MLP}_{\text{c\_proj}}$. It is important to note that, given the structure of factual knowledge, we only consider the last token for data extraction: the core idea is to test the reaction of the model to the concept.

During training, we collect and accumulate the gradients of the outputs and activations for the matrices $\text{Attn}_{\text{c\_attn}}$, $\text{Attn}_{\text{c\_proj}}$, $\text{MLP}_{\text{c\_fc}}$, and $\text{MLP}_{\text{c\_proj}}$. Let t denote the training step.

- **Activation Collection:**
  $\text{Attn}_{\text{c\_attn}}(t)$, $\text{Attn}_{\text{c\_proj}}(t)$, $\text{MLP}_{\text{c\_fc}}(t)$, $\text{MLP}_{\text{c\_proj}}(t)$
- **Gradient of the Outputs Collection:**
  $\nabla L\text{Attn}_{\text{c\_attn}}(t)$, $\nabla L\text{Attn}_{\text{c\_proj}}(t)$, $\nabla L\text{MLP}_{\text{c\_fc}}(t)$, $\nabla L\text{MLP}_{\text{c\_proj}}(t)$

We then perform a layer normalization for each layer l as follows:

$$\hat{A}_n^l = \frac{A_n^l - \mu_A^l}{\sigma_A^l}, \quad \hat{G}_n^l = \frac{G_n^l - \mu_G^l}{\sigma_G^l}$$

where μ_A^l and σ_A^l are the mean and standard deviation of activations in layer l, and μ_G^l and σ_G^l are the mean and standard deviation of gradients in layer l. The standardized metrics are then summed across the training steps to obtain historical activations and historical gradients:

$$H\hat{A}_n = \sum_{t=1}^{T} \hat{A}_n^l, \quad H\hat{G}_n = \sum_{t=1}^{T} \hat{G}_n^l$$

This historical data should allow us to determine whether neurons are"busy" (already containing and processing knowledge) or "free" (capable of encoding additional information).

3.3 Cognitive Awareness

Building upon the historical usage and change of neurons and weights, we implement a simple classifier that leverages both historical and current data to assess the nature of new information. Previous works [34, 35] studied ways for detecting when a model is hallucinating through the identification of patterns which prove the model is lying. While the task we aim to address is slightly different, it is closely related to hallucination detection. Instead of identifying patterns that reveal hallucinations, we seek to examine the model's internal response to external facts in order to gain insight into its internal beliefs. This step is fundamental to apply

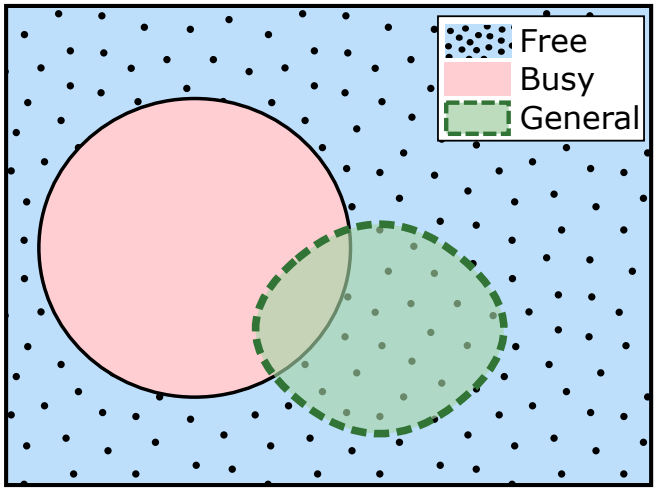

Figure 3.1: Graphical representation of the different type of neurons inside the model.

the proper update technique in the following step of the pipeline.

For a given input x (representing a new episode), we first perform a forward pass to obtain its current activations and a backward pass to obtain its current gradients (without updating the model weights). In particular, it takes as input the means and standard deviations of these current activations and gradients, as well as the historical activations $H\hat{A}$ and historical gradients $H\hat{G}$, accumulated as described in the previous section. Specifically, the classifier $\mathcal{C}$ is defined as:

$$\mathcal{C} : (x, \hat{A}, \hat{G}, H\hat{A}, H\hat{G}) \rightarrow \{\text{novel}, \text{familiar}, \text{conflicting}\}$$

Intuitively, the classifier should use the current state together with the historical patterns, to determine whether the new information is novel (and should be integrated), familiar (and can be ignored), or conflicting (and requires proper resolution). Considering all parameters from the model results in high-dimensional feature spaces, which can be challenging to manage it is necessary to take proper care of extracted features. A recently developed solution involves using sparse autoencoders [36] to address this issue. However, since the classifier needs to operate in an online environment, we opt for a shallow machine learning approach that leverages aggregated features. Specifically, in this work we focus on Support Vector Machine (SVM) and Random Forest (RF) classifiers. We extract statistical aggregates from each layer under analysis, allowing us to reduce dimensionality while retaining essential information for effective classification. For each layer we compute:

- Average
- Standard deviation
- Minimum
- First quartile
- Second quartile
- Third quartile
- Maximum

This aggregation method yields 7 features each for activations and gradients of a given layer, resulting in a total of 14 features per layer. This approach ensures scalability to larger models, as the feature dimensionality grows linearly with the number of layers rather than with the hidden dimensions. By focusing on layer-wise aggregation, this method maintains manageable feature complexity, making it suitable for increasingly complex models without succumbing to the exponential growth of high-dimensional spaces associated with deeper architectures.

3.4 Targeted Updates with Differentiated Plasticity

Building upon the historical tracking of neuron usage and the prior-belief awareness classifier, we implement targeted network updates to incorporate new knowledge or edit existing potentially conflicting information. This approach is based on the concept of Differentiated Plasticity, where neurons get updated differently depending on their knowledge content. We design three main types of targeted updates focusing on different types of neurons. Figure 3.1 illustrates the conceptual relationship between the different types of neurons within the model's parameter space. We define different types of neurons.

- **Free neurons:** neurons which do not contain previous knowledge.
- **Busy neurons:** neurons which contain previous knowledge.
- **General neurons:** neurons which are related to general factual knowledge capabilities.

The selection is based on the gradient accumulation observed during the learning of the previous knowledge, identifying those neurons which already incorporate information and those which are still free. Figure 3.2 represent the gradient accumulation distribution measured during a training run and it shows the busy and free areas.

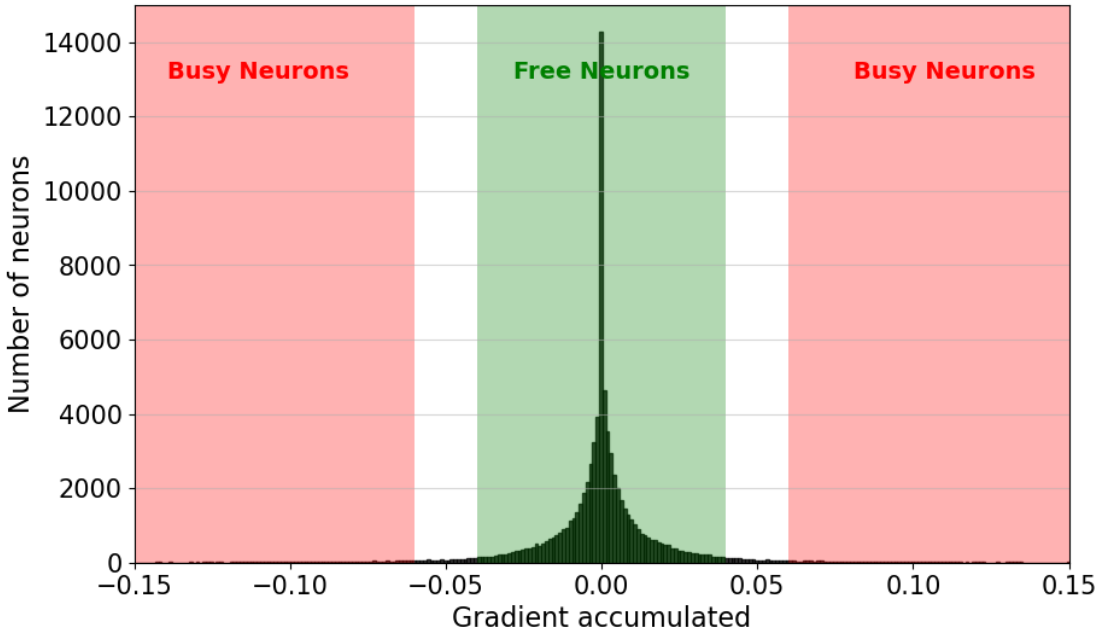

Figure 3.2: Distribution of the gradient accumulation of the neurons inside the model with graphical representation of free and busy areas.

Free Neurons Updates

In this strategy, we target neurons with low historical gradient value. To identify them, we rank neurons by increasing historical gradient values and select the top N neurons with the lowest cumulative gradients:

$$\mathcal{N}_{\text{free}} = \{n \mid \text{rank}(H\hat{G}_n) \leq N\},$$

where $H\hat{G}_n$ is the historical gradient for neuron n, accumulated over all prior training.

Rationale: By targeting underutilized neurons, we aim to incorporate new knowledge while minimizing interference with existing information. This strategy is particularly expected to be effective for incremental updates of novel, non-conflicting information.

Busy Neurons Updates

When targeting busy neurons, we search neurons that accumulated high historical gradients, indicating significant involvement in previous learning. Ranking neurons by increasing historical gradient values and select the top N neurons with the

highest values:

$$\mathcal{N}_{\text{stubborn}} = \{n \mid \text{rank}(H\hat{G}_n) > |\mathcal{N}| - N\},$$

where $|\mathcal{N}|$ is the total number of neurons, and $H\hat{G}_n$ is the historical gradient for neuron n. Updating stubborn neurons allows us to test the model's capacity for knowledge integration and assess the potential risks of overwriting existing information.

Rationale: Updating busy neurons allows us to investigate the model's capacity for knowledge integration and the potential risks of overwriting existing information. This approach is particularly relevant for editing existing knowledge, especially when dealing with conflicting information.

Specific Neurons Updates

Finally, we consider a hybrid solution. In this update, we target general neurons that are not part of the busy set; in other words, these neurons have general knowledge capabilities without impacting the specific facts that we want to protect. For reference, this set is represented by the green part, which does not intersect the red in Figure 3.1. The extraction process for these neurons is defined as follows:

1. We simulate a training step on a general set of facts.
2. We collect the gradient information from this simulated step.
3. We train the model on the first set of knowledge, the one we want to preserve.
4. During the targeted update phase, we identify the neurons with the highest accumulated gradients from the simulated training. From that set we exclude neurons that are among the top N busiest in relation to the knowledge to preserve. After applying this masking, we select the top N neurons with the highest gradient accumulation from the remaining set for the targeted update.

Rationale: This approach aims to test more precise updates by targeting general knowledge neurons which do not relate with previous knowledge, minimizing unintended changes. It is expected to be useful for both incremental updates and careful editing of existing knowledge.

Controlling Update Scope

Finally, for each update type, we vary the number of neurons involved (N) as an experimental parameter. This allows us to investigate the impact of update scope, from highly localized or sparse (small N) to more widespread (large N) updates.

Difference between Incremental and Counterfactual Updates

While we apply these targeted update strategies to both incremental updates (adding novel, non-conflicting information) and counterfactual updates (editing existing, potentially conflicting information), we have specific expectations. For incremental updates, we expect that targeting free neurons will be most effective, as it should minimize disruption to existing knowledge. For editing conflicting information, we expect that targeting busy or specific neurons will be necessary to modify the existing, incorrect knowledge effectively.

Chapter 4

Experimental setup

In this chapter, we detail the experimental setup used in our study, including the models, datasets, experiment details, and computational resources. Additionally, we provide a comprehensive account of all the specific parameters and configurations applied across our experiments to ensure transparency and facilitate the reproducibility of our results in future studies.

4.1 Models

The proposed approach is adaptable to models which have different internal structures. Considering the nature of our project and the available resources, we use as main model for our research GPT2-SMALL. The version we use can be found on *HuggingFace* [37] and it has the following characteristics:

- 117M parameters
- 12 layers
- 768 hidden dimension
- 12 attention heads

GPT2 has been released by OpenAI in 2019 in different versions as a natural improvement over the original GPT architecture. We use GPT2-SMALL, which represents a really light and portable solution with just 117 millions parameters. GPT2-SMALL is a good model for experimenting with new methods, since it has a very similar structure compared to bigger models but it requires lower computational resources. GPT2-XL keeps the same structure of the smaller version but it enlarges the hidden dimension and the number of blocks: future works can exploit this model, or even larger architectures such as GPT-J and Llama-7B, to conduct

research in more complex scenario. The proposed framework makes it possible by offering a complete modularity in terms of architecture.

4.2 Dataset

4.2.1 CounterFact

The CounterFact dataset [3] is an evaluation dataset designed to test counterfactual edits in language models, specifically aimed at distinguishing between superficial changes in model outputs and deeper, generalized modifications to factual knowledge. The dataset comprises 21,919 records, each constructed from knowledge tuples derived from PARAREL [38], which include subjects, relations, and objects, all existing as entities in WikiData [39]. Each item in the dataset is composed of different elements:

- *Facts*: These are the original true facts, which should be originally known by the model.
- *Counterfacts*: These are difficult false facts that start with low initial scores compared to their correct counterparts, used to challenge the model's ability to significantly alter its predictions.
- *Neighborhood prompts*: To test the locality, the dataset includes prompts involving subjects that are semantically related to the original ones but should remain unaffected by the edits.
- *Paraphrase prompts*: To assess generalization, a set of paraphrased prompts equivalent to the original counterfactual statements are provided.
- *Generation prompts*: These are used to evaluate the depth of the model's generalization by testing its ability to generate accurate outputs for broader, implicit queries related to the counterfactual facts. These prompts are hand-curated to draw out the underlying factual modifications from the model rather than direct queries.

4.2.2 Synthetic data generation

Part of our pipeline aims at classifying facts depending on the knowledge of the model. The CounterFact dataset offers in this sense an optimal base since it provides for each fact the original target (the real fact) and a new target which represent the new data to be learnt by the model (the counteract). Fine-tuning a model on the original facts can guarantee us that the model has knowledge about

them and we can perform analysis on the different model's response at coherent facts and counter-facts.

However, we are still missing the facts representing the third class that we want to analyze: the unfamiliar facts. Considering that small models are not accountable for general knowledge, we could simply extract a partition of the Counterfactual Dataset which is not included in the fine-tuning set and consider those facts as unknown. Nevertheless, we can not be scientifically sure that these data were not present in the original dataset used for the model pre-training. For this reasons, we opted for the generation of synthetic facts starting from the original data. For this purpose, we use a GPT-4 with a specific prompt to extract a fact with a similar structure with respect to the initial one, but fantasy names for keys and values. Manual checking of the facts is performed to verify the validity of the synthetic data: we want to avoid data which are too similar to the originals, since we want a measurable effect on the model but, at the same time, we want to keep meaningful facts with similar structure. Below an example of our synthetic generation pipeline:

Original: Danielle Darrieux's mother tongue is French
Transformed: Zorgon Flux's native language is Xylophian

For reproducibility purposes we insert the prompt used to generate the unknown facts. We generate each group of facts using batches of 25 elements each.

> Starting from this list of facts, can you create one data entry for each that concerns imaginary names and characters if necessary, while following the same logic.
> For example, Danielle Darrieux's mother tongue is French becomes Machin De Machine's mother tongue is Kurdi (or Kinduli). Edwin of Northumbria's religious values strongly emphasize Christianity becomes Hamed Habib's religious values strongly emphasize Atheism (or Peace or other values).
> Try to make the old and new as far as possible from each other (e.g., Kurdi is far from French, Kinduli is an imaginary language, etc.), while keeping some logic. Write in JSON format please (easy to parse).

4.2.3 Evaluation metrics

Taking inspiration from the literature [2], in this work we implement different metrics to evaluate the update and editing performances using the CounterFact dataset.

Update

- **Accuracy**: It is the ability to provide the correct completion given the original fact.

Edit

- **Reliability**: It is the ability to provide the correct completion given the edited fact.
- **Locality**: It measures the ability of applying specific edits to the model, leaving unaffected unrelated facts. Given the incremental nature of our work, we do not make use of neighborhood prompts, but we measure the locality as the accuracy on the previous knowledge which should not be affected by the edit.
- **Generalization**: To verify the actual ability of the model to learn general concepts rather then specific facts, we exploit the generation prompts. These prompts are built to offer more difficult relations compared to the classical paraphrase prompts: in this way we can truly stress the capacity of our model.

4.3 Implementation details

4.3.1 Fine-tuning process

For running our experiments we need to be sure about the model's knowledge related to the facts we consider *coherent*: this aspect is crucial for performing consistent and meaningful analysis of the internal parameters of the model. To build this class we select as our ground truth the true facts from the Counterfact dataset: they represent true statement in the form of key-value relations. However, considering we do not work with models big enough and we have no direct access to the original training data we cannot be sure that some specific facts are already included in the model knowledge. As standard way for ensuring the model's factual knowledge related to our data we perform a fine-tuning procedure using the true facts.

As additional constraint we would like to verify that all the facts have been correctly learnt by the model, since it is not guaranteed that all the fine-tuning set is successfully inserted into the model's knowledge. In order to reach this goal we fine-tune the model for a number of epochs big enough to guarantee 100% accuracy over the training set: in this manner we are sure that the model learnt all the facts.

Instead, for what concerns the fine-tuning process itself, given the autoregressive nature of GPT models, we apply a masked CLM computing the loss only on the answer tokens. This implementation guarantees that the model is trained solely on providing the correct answer given the context, rather than learning the entire sentence structure.

In terms of implementation we use as baseline implementation the one proposed by EasyEdit [40] in order to be consistent with results obtained in previous works. The process is based on an auto-regressive training of the model where the entire sentence, composed by both prompt and completion, is given as input to the model. The expected output is the sentence itself: in this way the model is able to learn how to answer token by token. To do so it is crucial to apply a shift in the labels: starting from the input sequence, we shift the tokens to the right not considering the first one. As a matter of fact, it will be a [START] token used to start a new generation; the logits instead will start from the second token (which is the first actually shown in the output).

4.3.2 Familiarity and dissonance detection details

For the familiarity and dissonance detection task we make use of two different classifiers, a SVM and a Random Forest. In order to provide robust results the results from each experiment are obtained by a 5-folds nested cross validation process: we apply this technique in order to provide statistically meaningful results. We perform an hyper-parameter tuning for each classifier using grid search on the following parameters:

SVM

- *C*: 0.1, 1, 10, 100
- *gamma*: scale, 1, 0.1, 0.01, 0.001
- *kernel*: linear, poly, rbf, sigmoid
- *degree*: 2, 3, 4
- *coef0*: 0, 0.1, 0.5, 1

Random Forest

- *n_estimators*: 50, 100, 200, 300
- *max_depth*: None, 10, 20, 30
- *min_samples_split*: 2, 5, 10

- *min_sample_leaf*: 1, 2, 4
- *bootstrap*: True, False

4.3.3 Knowledge update details

In order to provide meaningful results each experiment has been tested on 5 different folds of the dataset. We used the same hyper-parameter configurations for the fine-tuning across the different update experiments since we wanted to keep consistency across the different settings. For the general knowledge neurons extraction we used 5 epochs with learning rate equal to 5×10^{-4}. For the previous knowledge and the new knowledge instead, we used learning rate equal to 10^{-3} and 10 epochs. The batch size was fixed at 32 for all the training procedures.

4.3.4 Knowledge edit details

As for the editing experiments we kept similar configurations compared to the knowledge update: 5 epochs and learning rate 5×10^{-4} for general knowledge neurons identification and 10 epochs with learning rate 10^{-3} for previous knowledge and new knowledge. The only difference is the addition of the editing step, which keep the 10^{-3} learning rate, but it uses 20 epochs: this change was done since the edit requires more iterations to reach a good reliability. The batch size, as before, was fixed at 32 for all the training procedures.

4.4 Computational needs

In terms of computational requirements, our proposed approach demands similar resources to those needed for full fine-tuning of language models. When working with a smaller model like GPT-2 SMALL, all experiments can be conducted on a single Nvidia V100 equivalent GPU, which provides sufficient computational power for this scale of model. To further enhance training speed and efficiency, parallel GPU configurations can be implemented using DeepSpeed [41], which optimizes distributed training processes and can significantly reduce the time required for experiments. For scaling this approach to larger models, such as GPT-J or LLaMA-7B, more robust computational resources would be necessary due to the increased number of parameters and higher memory demands. A widely adopted choice in the literature is to use Nvidia A100 GPUs with 80GB of memory.

Chapter 5

Experiments and results

In this chapter, we present the experiments conducted to evaluate the effectiveness of our method. The experiments are organized into three distinct categories: familiarity and dissonance detection, incremental knowledge update, and incremental knowledge editing. Each of these categories aims to verify different areas of our new learning paradigm in controlled settings, to have a complete view of the capabilities of the proposed method. For each category, we provide a comprehensive description of the experimental procedures, followed by a presentation of the results, along with detailed analysis and explanations. Each main experiment is complemented by ablation studies to provide a more comprehensive evaluation and to validate the robustness of our approach. The implementation details for each experiment are thoroughly outlined in Chapter 4.

5.1 Familiarity and dissonance detection

5.1.1 Detection on fine-tuned model

Description

The goal of this experiment is to demonstrate that a cognitive dissonance metric can be established using the gradient and activation response of a model. We begin by selecting a pre-trained model and fine-tuning it on a set of 1000 facts to ensure that it has fully internalized this knowledge. These facts will serve as a baseline for the experiment. Then, we need to create a labeled dataset which includes three classes of facts:

- *Familiar Facts*: These are the facts the model was fine-tuned on. To ensure generalization, we will present these facts using various rephrased prompts.

- *Dissonant Facts*: These are statements that directly contradict the known facts, designed to assess the model's response to conflicting information.
- *Unknown Facts*: These are facts generated using a prompted GPT-4 model (Chapter 4, ensuring that the data points are novel and not inherently known or counter to the facts the model was trained on.

In total we now have a balanced dataset composed of 3000 facts. With this data, we train two machine learning classifiers: a Support Vector Machine (SVM) and a Random Forest. The objective is to enable these classifiers to distinguish between coherent, dissonant, and unfamiliar facts based on their familiarity and dissonance in relation to the model's internalized knowledge. By evaluating the classifiers' ability to correctly categorize these classes we aim to quantify the cognitive dissonance of the model, effectively measuring how the model's gradient and activation responses vary in relation to the familiarity and consistency of the input with its trained knowledge.

Results

Normalization	Classifier	Accuracy	F1 Score
Null	SVM	0.994 (0.004)	0.994 (0.004)
	RF	0.988 (0.001)	0.988 (0.001)
Layer	SVM	0.995 (0.001)	0.995 (0.001)
	RF	0.982 (0.005)	0.982 (0.004)
Historical	SVM	0.995 (0.001)	0.995 (0.001)
	RF	0.978 (0.003)	0.978 (0.003)

Table 5.1: Results of familiarity and dissonance detection experiment on facts known by the fine-tuned model.

The classification task achieved an accuracy exceeding 99%, with results consistently stable across various normalization methods. Both classifiers, the SVM and Random Forest, performed similarly, although the SVM showed a slight advantage. Examining the confusion matrix (Figure 5.1) reveals that the classifiers successfully learned to distinguish between familiar and dissonant facts. However, a minor error persists when classifying unknown facts. Feature importance analysis indicates that the gradient serves as the primary discriminative factor (Figure 5.2), with transformer block 4 emerging as the most significant contributor to the classification

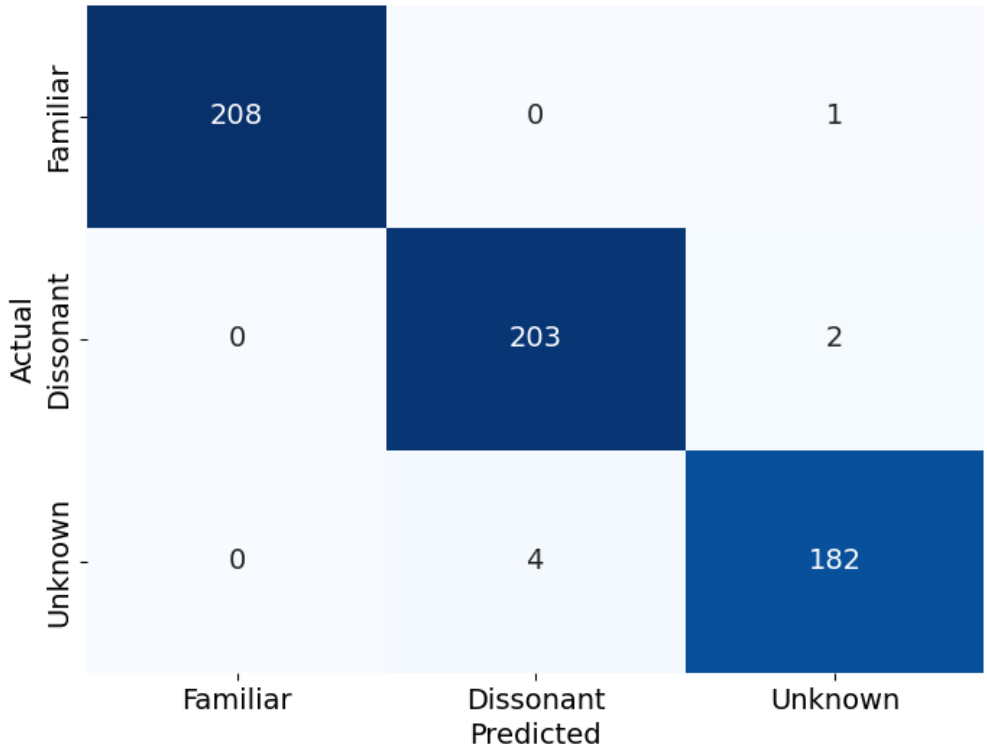

Figure 5.1: Confusion matrix related to the familiarity and dissonance detection on the fine-tuned model.

task (Figure 5.3). The importance of the gradient was expected, since the model is fine-tuned on the specific facts we are using for the evaluation: for this reason the expected loss and the related gradient generated by the backpropagation should be almost null.

While these results are promising, it is important to note that these metrics were obtained from a model fine-tuned specifically on the facts in question. This scenario raises concerns about the integrity of our experiment, as there is a potential risk of overfitting. These concerns rises in particular observing the major importance shown by the gradients, which could suggest that the model learnt by heart the facts without learning the actual concepts. To validate the robustness and efficacy of this method, it is crucial to conduct an ablation study using a pre-trained model with a less peaked response. This approach will help determine whether the high accuracy is genuinely due to the model's ability to differentiate between familiar, dissonant, and unknown facts in a real-world scenario.

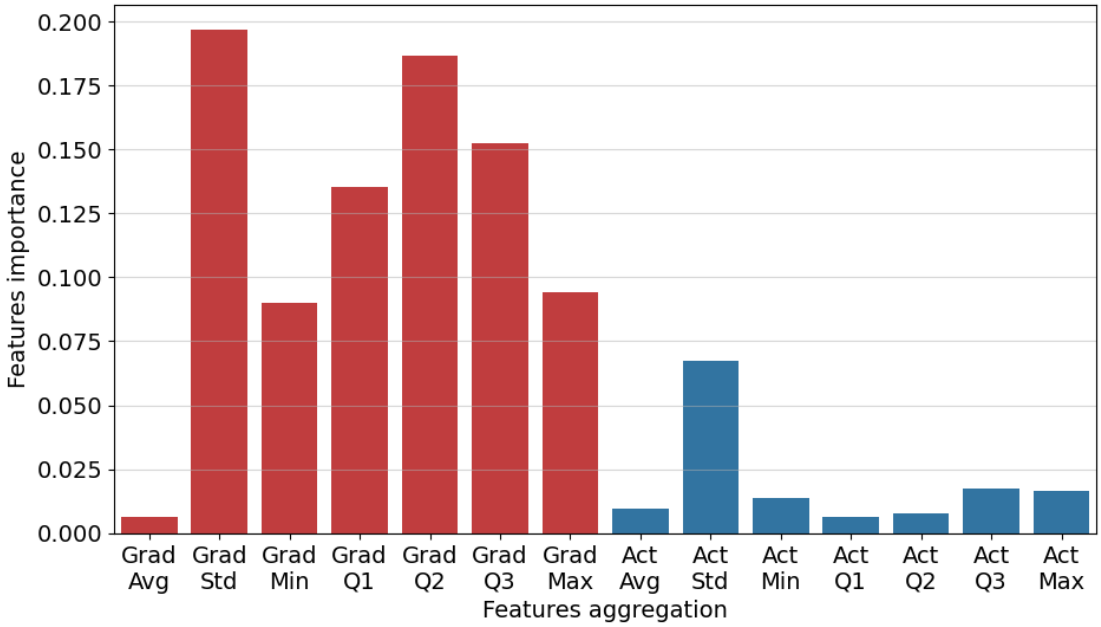

Figure 5.2: Comparison between the features importance of the different types of aggregation methods in the fine-tuned model.

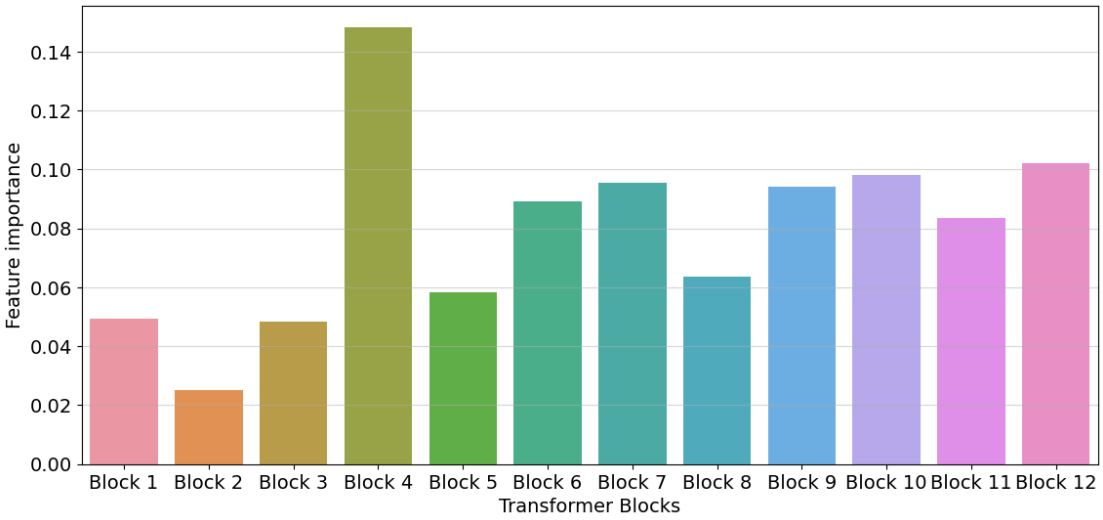

Figure 5.3: Comparison between the features importance of the different blocks in the fine-tuned model.

5.1.2 Detection on pre-trained model

Description

To validate the efficacy of our method in a real-world scenario, we decided to test its detection capabilities on a pre-trained model, without additional fine-tuning on the specific set of facts. To construct a set of known facts, we conducted an inference round using the entire dataset, identifying the facts for which the pre-trained model could provide correct responses. This approach allowed us to create a classification dataset consisting of facts that the model already recognized from its pre-training phase.

Results

Normalization	Classifier	Accuracy	F1 Score
Null	SVM	0.944 (0.006)	0.944 (0.006)
	RF	0.928 (0.012)	0.929 (0.011)
Layer	SVM	0.949 (0.006)	0.949 (0.006)
	RF	0.909 (0.014)	0.910 (0.013)
Historical	SVM	0.947 (0.004)	0.948 (0.003)
	RF	0.925 (0.006)	0.925 (0.006)

Table 5.2: Results of familiarity and dissonance detection experiment on facts known by the pre-trained model.

The experiment overall produced positive results (Table 5.2), since we were able to reach a peak performance close to 0.95 in accuracy, which is slightly less compared to the classification on the fine-tuned model. In this experiment too, the normalization does not seem to have a visible impact on the results. Analyzing the confusion matrix (Figure 5.4), we notice that the error is mostly due to the confusion between familiar and dissonant labels. It is worth noticing that, the features importance distribution is different compared to the classification on the fine-tune model. Looking at Figure 5.5, it is clear that the activations play a more important role: this behavior was expected, since in a fine-tuned there is an almost null gradient for all the facts which the model has been trained on. Also looking at the results of the single modality, it is noticeable how activations-only experiments provide results in line with those with the gradients. Instead of having a predominant block in terms of importance, in this situation the importance across

blocks (Figure 5.6) seems equally distributed.

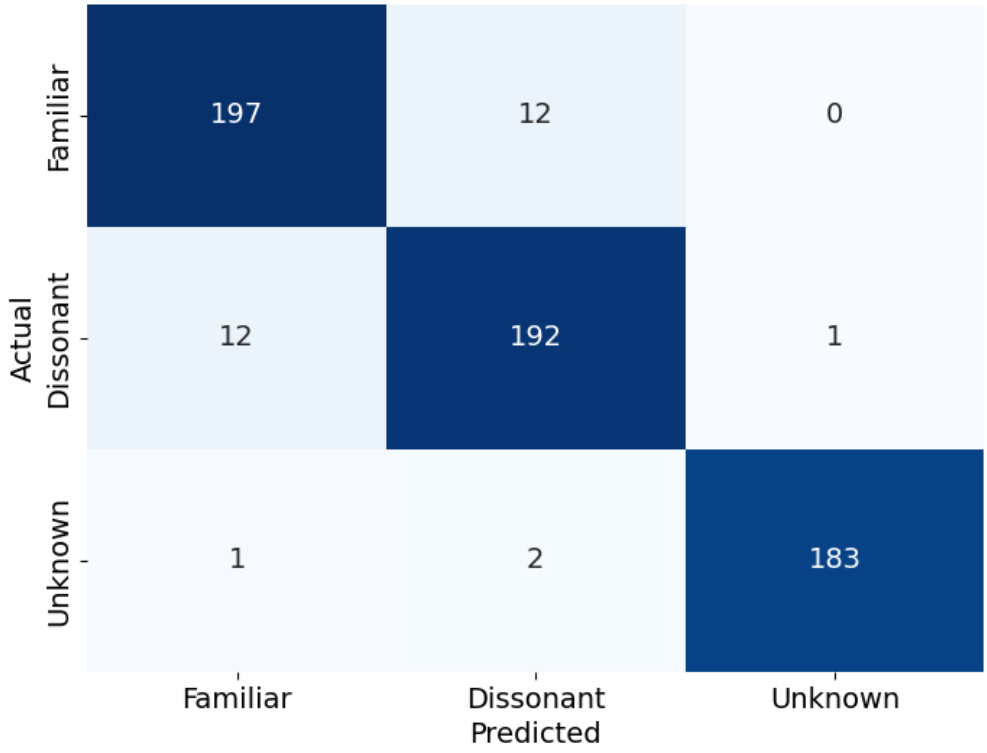

Figure 5.4: Confusion matrix related to the familiarity and dissonance detection on the pre-trained model.

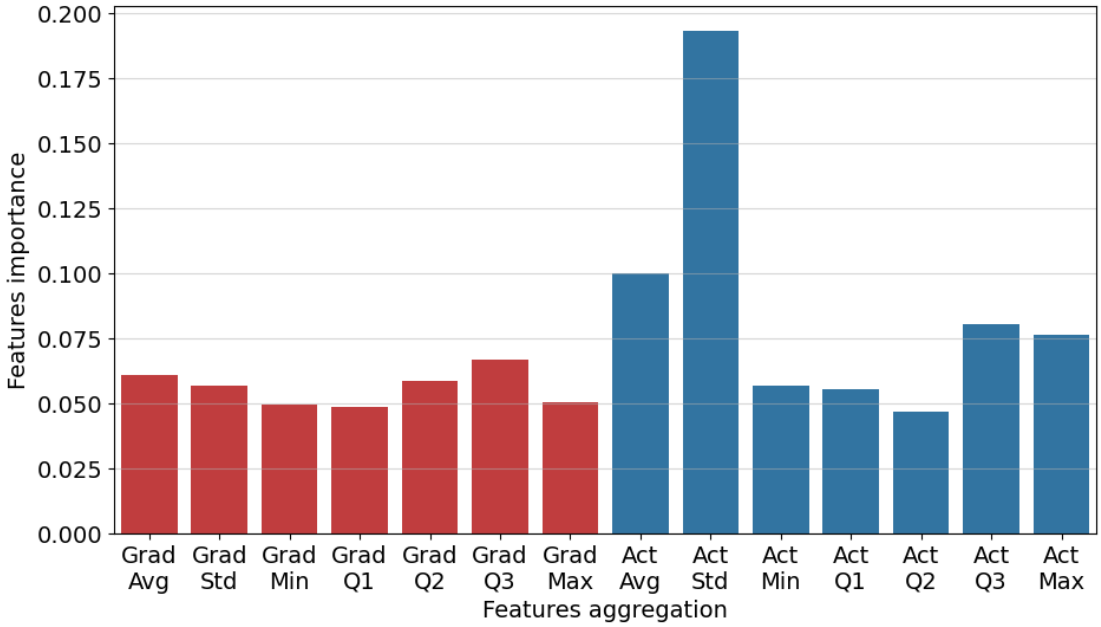

Figure 5.5: Comparison between the features importance of the different types of aggregation methods in the pre-trained model.

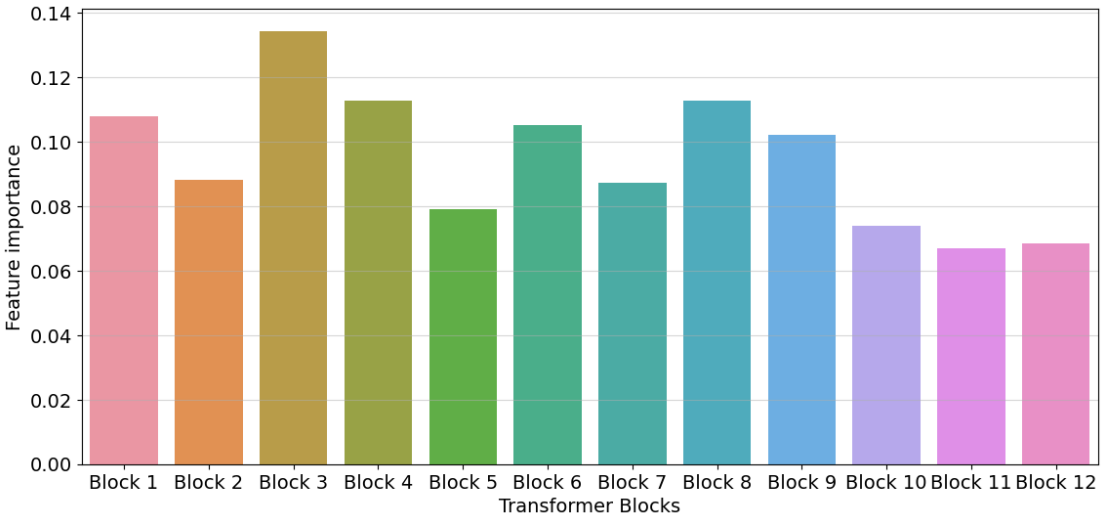

Figure 5.6: Comparison between the features importance of the different blocks in the pre-trained model.

5.2 Incremental update

5.2.1 Incremental knowledge update

Description

To evaluate the model's ability to retain previously learned knowledge while integrating new information, we designed an experiment that measures accuracy on existing facts after incorporating additional knowledge. The process is divided into different steps. As first thing, we simulate a fine-tuning on 10000 facts on the pre-trained model keeping track of the gradient accumulation. This step is fundamental for detecting the general knowledge neurons. Then, starting from the pre-trained model we perform a fine-tuning on a set of 2000 facts, which represent the core knowledge that needs to be preserved. During this phase, we track the accumulation of gradients, which allows us to construct the HAF necessary for the subsequent targeted neuron updates. Finally, we introduce an additional 1000 facts to the model, building upon the initial knowledge base testing different types of targeted updates, comparing them to the full tine-tuning. We measure the accuracy on both previous and new knowledge, to assess the knowledge preservation capabilities. The different approaches we compare are:

- Full fine-tuning
- Targeted FT on Busy neurons
- Targeted FT on Free neurons
- Targeted FT on Specific neurons
- Targeted FT on Random neurons

Results

Figure 5.7 shows good results in terms of knowledge preservation: all our sparse training approaches outperform full fine-tuning, providing a significant advantage. The first observation we can make is that modifying fewer neurons generally results in less disruption to the existing knowledge. Among the methods evaluated, utilizing free neurons proves to be the most effective for integrating new knowledge, underscoring the efficacy of our proposed method. In contrast, altering busy areas results in greater disruption, as these regions are more influenced by previous training. Specific neurons present an intermediate performance between busy and free neurons, which aligns with expectations, given that these neurons were intended to modify general neurons not explicitly tied to prior knowledge. Randomly selected neurons also exhibit highly promising results, nearly preserving previous knowledge

entirely, with performance comparable to the free selection method. This approach benefits from the inherent advantages of sparsity: modifying fewer neurons, regardless of their type, generally reduces the impact on the output. Moreover, in a network that lacks dense knowledge packing, altering random parameters is less likely to cause significant disruption.

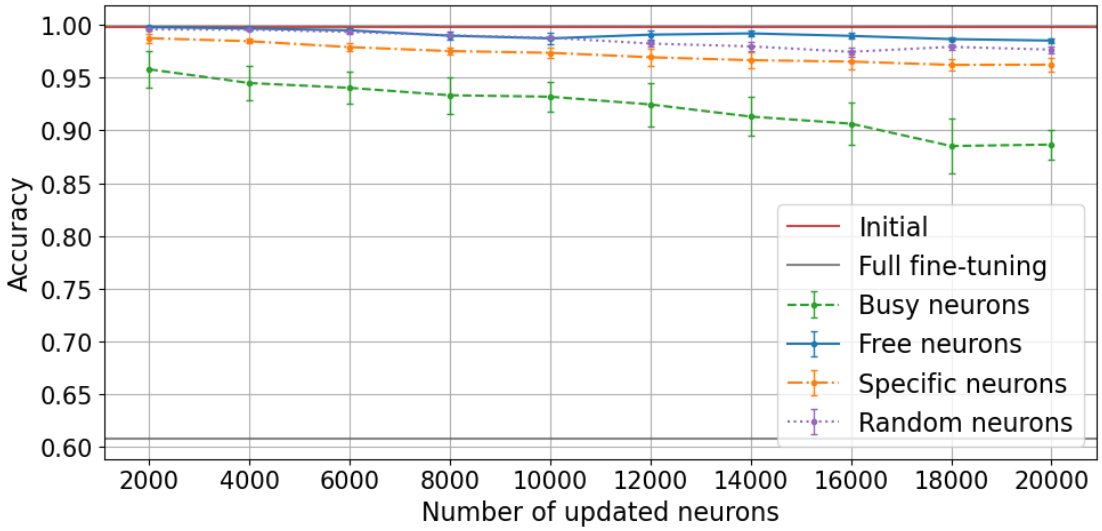

Figure 5.7: Accuracy score on the set of previous knowledge comparing different numbers and types of neurons.

When analyzing the accuracy on the new knowledge (Figure 5.8), it is evident that busy neurons have a superior capacity for learning new information compared to free neurons. Busy neurons exhibit a higher knowledge density, as they can store more information within the same update area size compared to other neuron types. By concentrating the updates on busy neurons, the same accuracy level as full fine-tuning can be achieved by modifying only 4000 neurons. In comparison, free neurons require adjustments of approximately 20000 to reach similar results. Specific neurons perform similarly to busy neurons but with a slightly lower knowledge density. Although random selection proved effective in preserving old knowledge, it does not offer sufficient knowledge density when compared to busy and specific neurons.

After the first fine-tuning, we obtain a model that incorporates the desired knowledge. Figure 5.9 presents the accumulation distribution of the neurons in the network, showing that most neurons exhibit negligible changes in their parameters throughout the training process. However, some neurons are noticeably impacted,

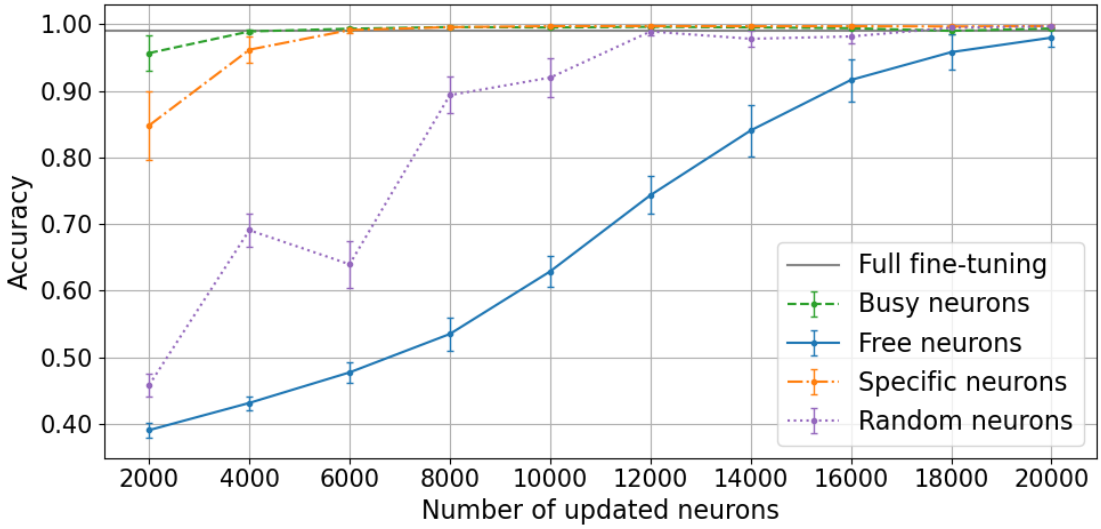

Figure 5.8: Accuracy score on the set of new knowledge comparing different numbers and types of neurons.

highlighting a sparse situation with only a few busy neurons in the network compared to the large number of neurons that are not significantly affected by the training.

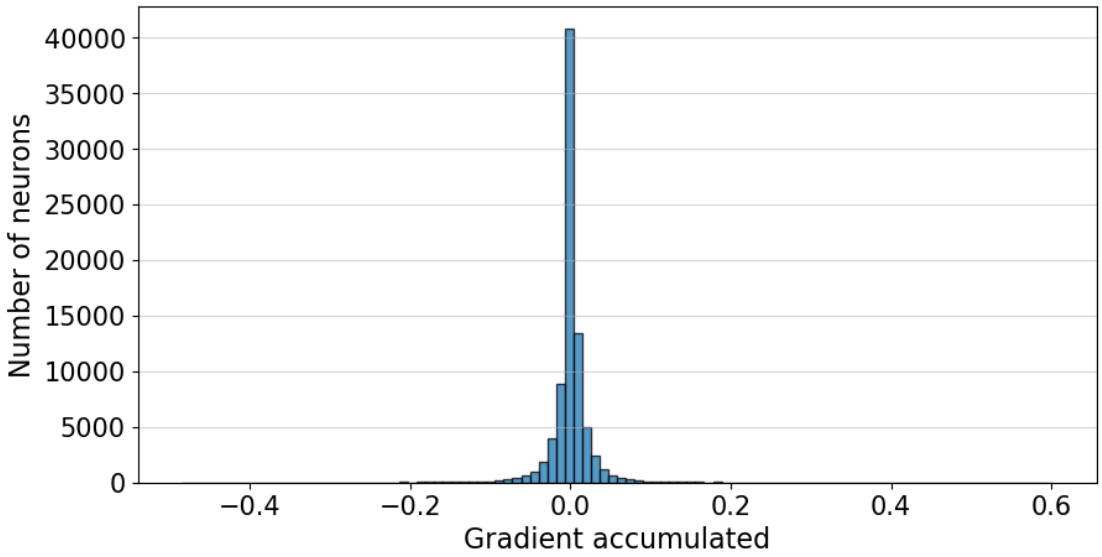

Figure 5.9: Gradient accumulation measured during the full fine-tuning related to the previous knowledge.

When showing the percentage of busy neurons for each layer, it is clear that there is strong concentration of them in the projection layer of the attention of the first transformer block (Figure 5.10). The general trend for all the blocks instead, is to have an higher percentage of busy neurons in the cross attention layer compared to the others. The MLP layers are the less impacted: this situation opposes previous work, which tends to focus the update on the fully connected layers.

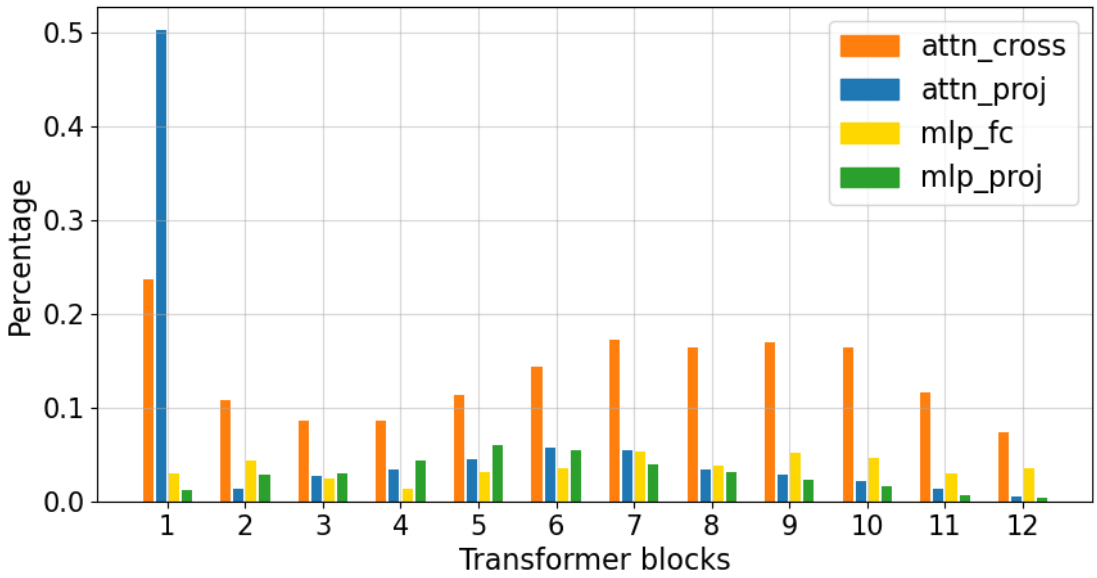

Figure 5.10: Percentage of neurons considered busy for each layer when selecting a total of 6000 busy neurons.

Despite the good performances compared to a full fine-tuning, we are not able to protect entirely the previous knowledge. This phenomenon could be due mainly to two elements: inefficacy of the location method and impact of the modifications on the computational graph. The location inefficacy could be a possible cause of knowledge degradation since we select the neurons to be updated using a heuristic: however the original knowledge was injected inside the model with a full fine-tuning, so the neurons free according to our definition could still be responsible for the previous knowledge to some extent. The latter instead derives from the fact that, even though we are trying to separate the areas we are working on, the neurons will be always connected to each other: this means that modifications done in different areas could still impact the propagation of the signal inside the network modifying the output.

For what concerns the higher knowledge density showed by busy neurons, they

can represent a lottery ticket configuration, and so they could be defined as a set of neurons accountable for learning general knowledge. However, this phenomenon could be just related to the fact that those neurons have been already trained on factual data and in some way they adapted at learning factual knowledge.

5.2.2 Local fine-tuning using lottery ticket configurations

Description

The results from our previous experiment support the lottery ticket hypothesis. To further demonstrate this analogy, we designed a new experiment to show that our selection of busy neurons can be seen as a lottery ticket configuration. In this new setup, we first fine-tune on 10000 facts and identify the busy neurons from this process. We then use this specific set of neurons to learn 2000 new facts, starting from a pre-trained model. For comparison, we conduct this procedure using different configurations of neurons:

- Full fine-tuning
- Targeted FT on Busy neurons
- Targeted FT on Free neurons
- Targeted FT on Random neurons

Results

It is noticeable that busy neurons are those providing the highest knowledge density also in this condition, as depicted in Figure 5.11. These results suggest a perfect analogy with the lottery ticket theory, excluding the potential effect of previous training.

Moreover, busy configurations are not just more dense but they are also faster at learning: Figure 5.12 shows the accuracy of the different selections across the epochs. Considering the same amount of neurons to update, the free set needs more iterations to reach the same accuracy as the busy set. The full fine-tuning is still faster at learning reaching a plateau before than the sparse selections.

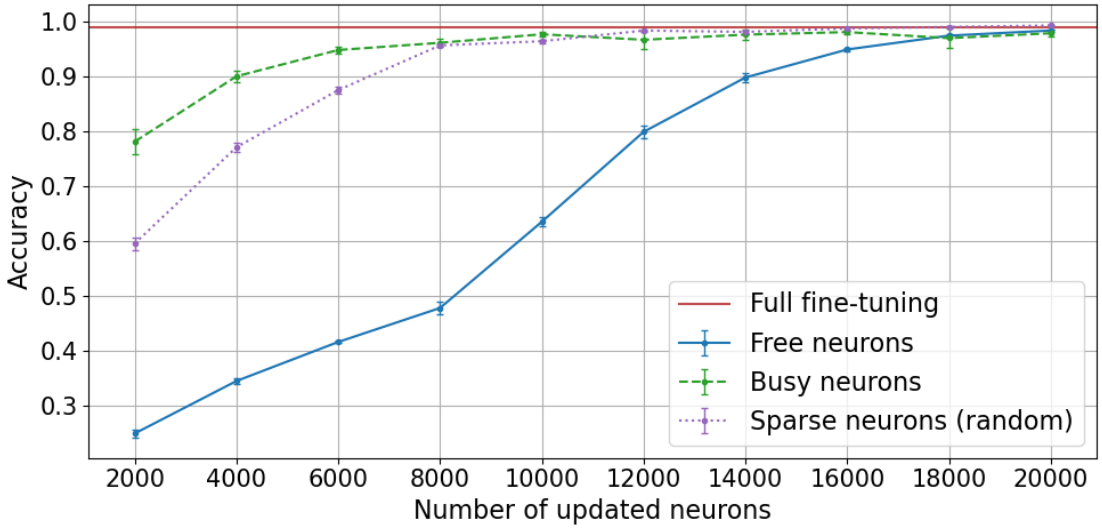

Figure 5.11: Accuracy score on the set of new knowledge inserted using lottery ticket configuration, comparing different numbers and types of neurons.

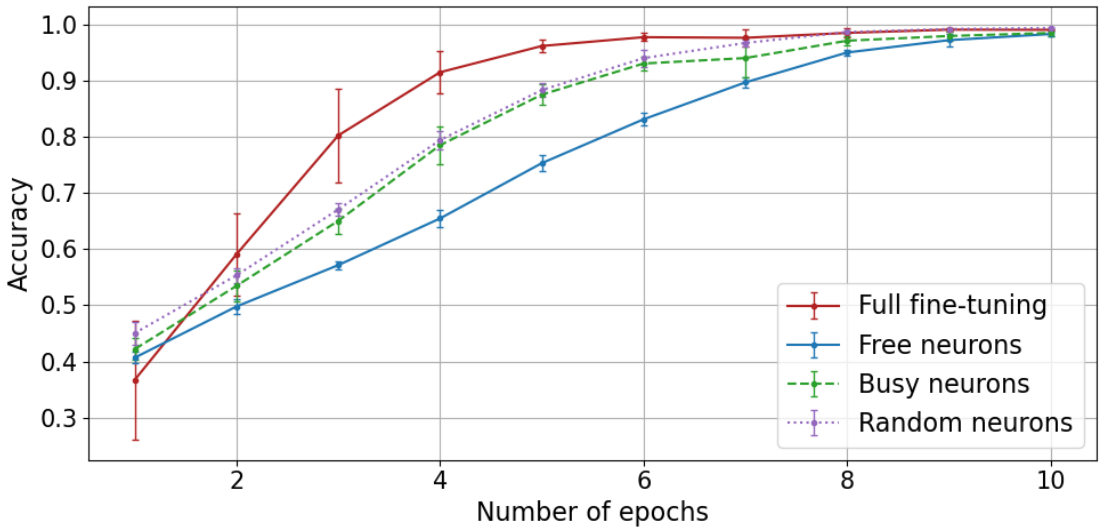

Figure 5.12: Accuracy score on the set of new knowledge inserted using lottery ticket configuration across the different epochs, updating 20000 neurons and comparing different types of selections.

5.2.3 Incremental update with sparsity by design

Description

To assess the impact of partial overlap in knowledge due to imperfect localization, we conduct an experiment where the knowledge is intentionally divided to enforce sparsity by design. In this setup, we ensure that there is no overlap between the two update regions: the model is updated in two steps, carefully avoiding any contamination between the areas. The update areas are selected using a lottery ticket configuration derived from a prior training session. Specifically, we simulate a training procedure with 10000 facts extracting the gradient accumulation. Then, the first set of knowledge composed by 2000 facts is inserted into the network freezing the top 20000 neurons: this technique allow us to keep a complete disjoint area for future updates. Finally, we incorporate 1000 new facts in the free area, always using different number of neurons. This approach allows us to directly compare the accuracy of our localization method against a genuinely sparse condition.

Results

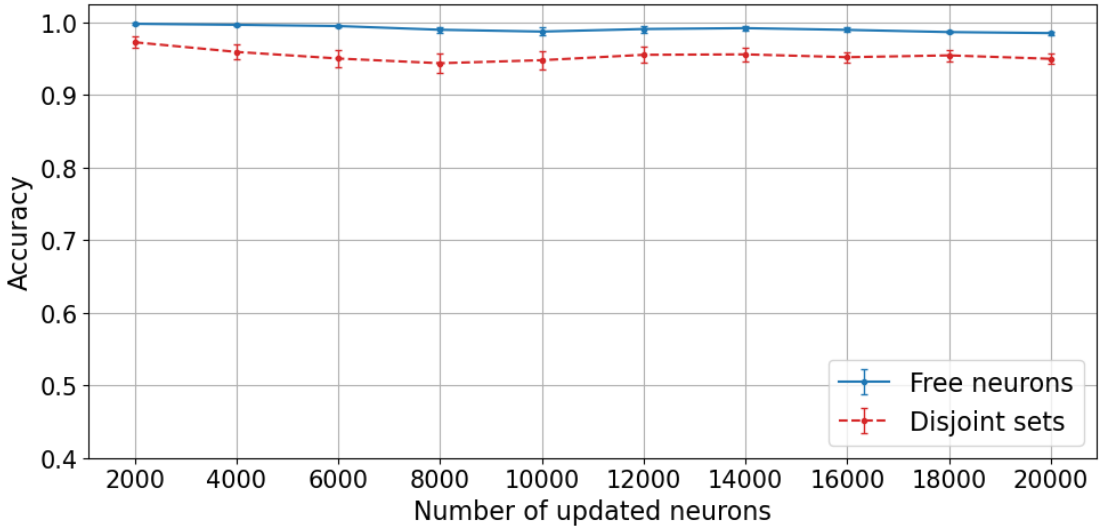

Figure 5.13: Accuracy score on the set of previous knowledge. Comparison between free neurons update and forced sparsity.

The results of the experiment show that even with forced sparsity it is not possible to preserve completely the previous knowledge when performing an update (Figure 5.13. Moreover, it is possible to notice that the results from the original experiment are even better when we consider the ability to preserve knowledge. The gap between the two experiments increases when we increase the number

of updated neurons: this behavior can be justified by the fact that, in order to guarantee the absence of overlapping, when we increase the number of neurons targeted for the update we reduce the number of neurons allocated for previous knowledge. This experiment proves that when performing knowledge update there is an impact related to the influence of the computational graph. On the contrary, the disjoint neurons perform better when learning new knowledge (Figure **??**).

The results of the experiment indicate that, even with enforced sparsity, it is not possible to completely preserve the previous knowledge during an update (Figure 5.13). Interestingly, the findings from the original experiment demonstrate a superior ability to preserve knowledge compared to this forced sparsity scenario. The gap between the two experiments widens as the number of updated neurons increases, which can be explained by the trade-off between the absence of overlap and the reduced allocation of neurons for previous knowledge. This experiment highlights that updating knowledge impacts the preservation of previous information due to the underlying influence of the computational graph. In contrast, disjoint neurons show better performance in learning new knowledge (Figure 5.14), suggesting that separate neural allocations are advantageous when introducing new information.

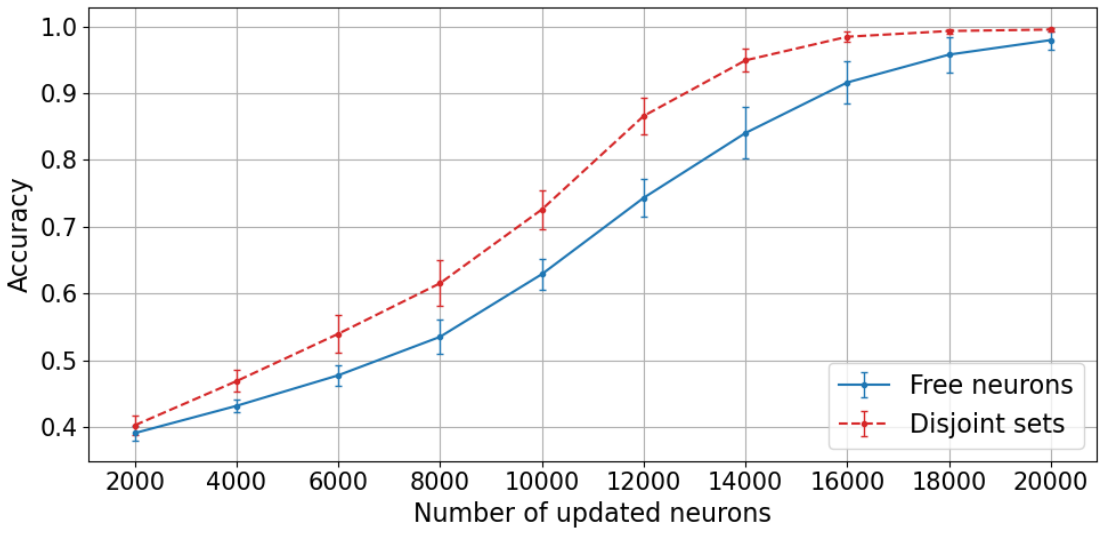

Figure 5.14: Accuracy score on the set of new knowledge. Comparison between free neurons update and forced sparsity.

5.3 Incremental editing

Description

After demonstrating the potential of sparse fine-tuning for knowledge insertion, we shift our focus to knowledge editing, which represents the final stage in our comprehensive pipeline. Unlike training to acquire new knowledge, knowledge editing aims to modify the model's existing beliefs. The procedure for this experiment involves several key steps: we begin by fine-tuning the model on a dataset containing 2000 facts, establishing the baseline knowledge set. A second round of fine-tuning is then performed, introducing a new set of facts (composed of 10, 100 or 1,000 elements) which serve as the targets for our subsequent editing process. To accurately measure the impact of the second fine-tuning on the original knowledge, we identify which facts from the initial dataset the model still recognizes, using this subset as a metric for evaluating the locality of changes made during the editing process. The final step is to edit the model's beliefs specifically regarding the facts introduced during the second fine-tuning. The objective is to update these beliefs while preserving the integrity of the previously established knowledge. We also evaluate the generalization capabilities of the editing by measuring accuracy on paraphrased versions of the edited facts. The different targeted strategies are tested against the full fine-tuning as baseline and ROME, which represents the current state-of-the-art.

Results

From the results presented in Table 5.3, it is evident that editing an existing concept is generally more challenging than incorporating new knowledge. Starting from the baseline established by full fine-tuning, we observe that while this approach allows for the effective learning of new information, it significantly disrupts the retention of prior knowledge, particularly when compared to the results of knowledge update experiments. Nonetheless, full fine-tuning proves to be the most versatile method in terms of generality. Regarding the sparse update method, a key finding is the limited effectiveness of using free neurons for editing facts, especially when compared to the busy neuron selection approach. This result confirm the initial hypothesis. Additionally, the ROME method exhibits notably low performance, which aligns with previous literature [2] demonstrating its inability to maintain performance at scale.

Upon detailed analysis of the relationship between different metrics and the number of neurons, we observe that the reliability (Figure 5.15) shows patterns similar to those seen during the update phase in Experiment 5.2. This indicates that the capability to modify existing knowledge remains intact. However, a significant

Method	Edited facts	Reliability	Locality	Generality
Full FT	10	1.000 (0.000)	0.107 (0.082)	0.576 (0.117)
	100	0.998 (0.003)	0.238 (0.019)	0.434 (0.089)
	1000	0.991 (0.009)	0.182 (0.007)	0.442 (0.053)
Free neurons	10	0.000 (0.000)	0.936 (0.029)	0.000 (0.00)
	100	0.033 (0.011)	0.553 (0.056)	0.016 (0.010)
	1000	0.247 (0.028)	0.266 (0.034)	0.060 (0.011)
Busy neurons	10	0.988 (0.024)	0.596 (0.106)	0.644 (0.128)
	100	0.969 (0.033)	0.542 (0.035)	0.462 (0.081)
	1000	0.996 (0.002)	0.199 (0.014)	0.380 (0.041)
Random neurons	10	0.380 (0.132)	0.827 (0.083)	0.092 (0.094)
	100	0.969 (0.033)	0.508 (0.019)	0.462 (0.081)
	1000	0.784 (0.091)	0.159 (0.032)	0.102 (0.014)
Specific neurons	10	0.964 (0.039)	0.638 (0.138)	0.512 (0.238)
	100	0.760 (0.063)	0.531 (0.019)	0.263 (0.027)
	1000	0.993 (0.003)	0.240 (0.017)	0.287 (0.039)
ROME	10	0.240 (0.162)	0.891 (0.075)	0.180 (0.160)
	100	0.300 (0.048)	0.430 (0.096)	0.150 (0.032)
	1000	0.160 (0.082)	0.152 (0.063)	0.067 (0.030)

Table 5.3: Results of the editing process considering different number of facts to be edited. In this table, the targeted update is performed using 8000 neurons.

limitation of this method is evident in terms of locality (Figure 5.16): performance on previously acquired knowledge deteriorates as early as when using 2000 neurons, particularly with busy and specific neurons. For a comprehensive view, we also include the generalization metrics in Figure 5.17. Notably, the generalization results remain stable regardless of the number of neurons modified, suggesting that the limitation is not inherently tied to the number of neurons used, especially for busy and specific neurons.

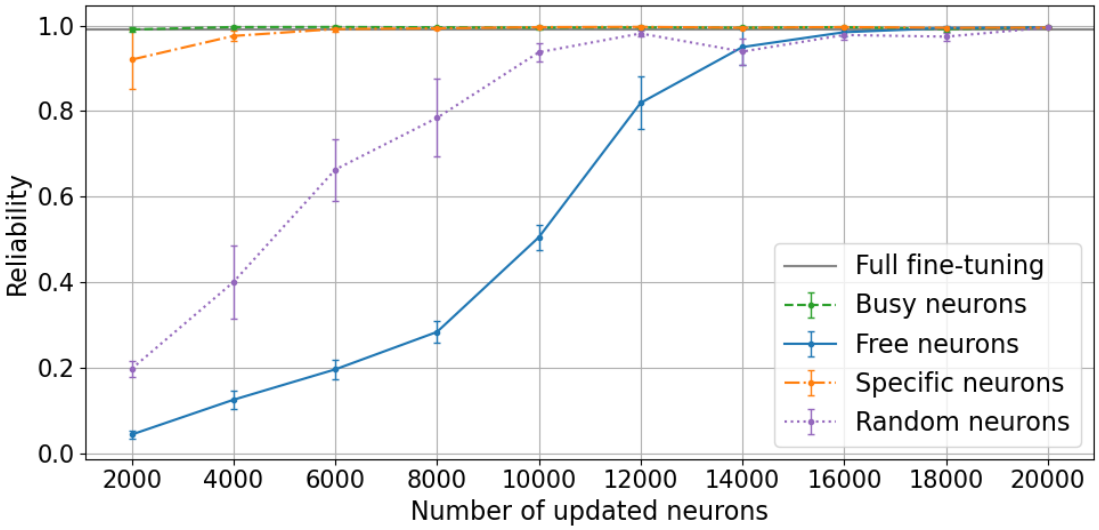

Figure 5.15: Comparison between the reliability score of different techniques when editing 1000 facts.

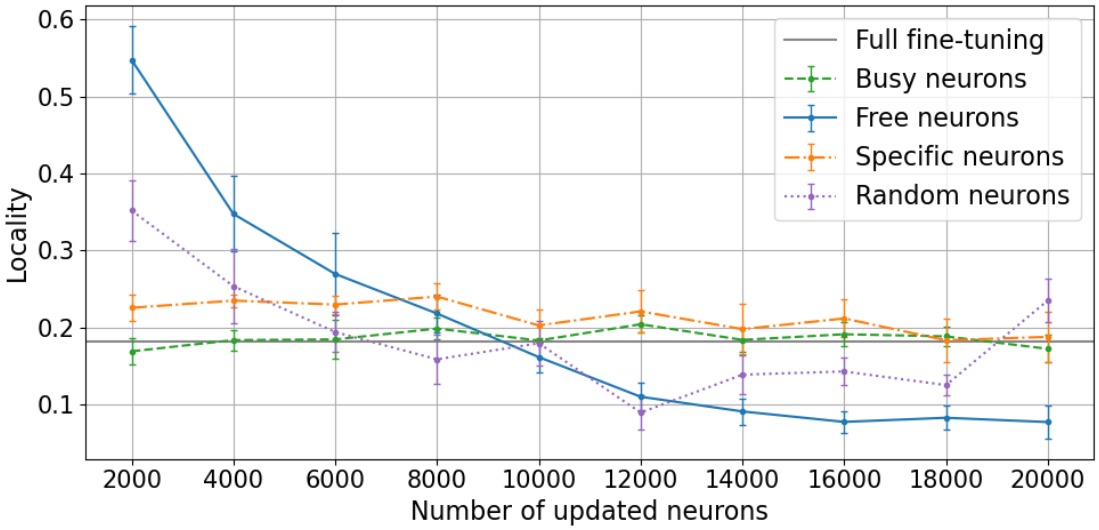

Figure 5.16: Comparison between the locality score of different techniques when editing 1000 facts.

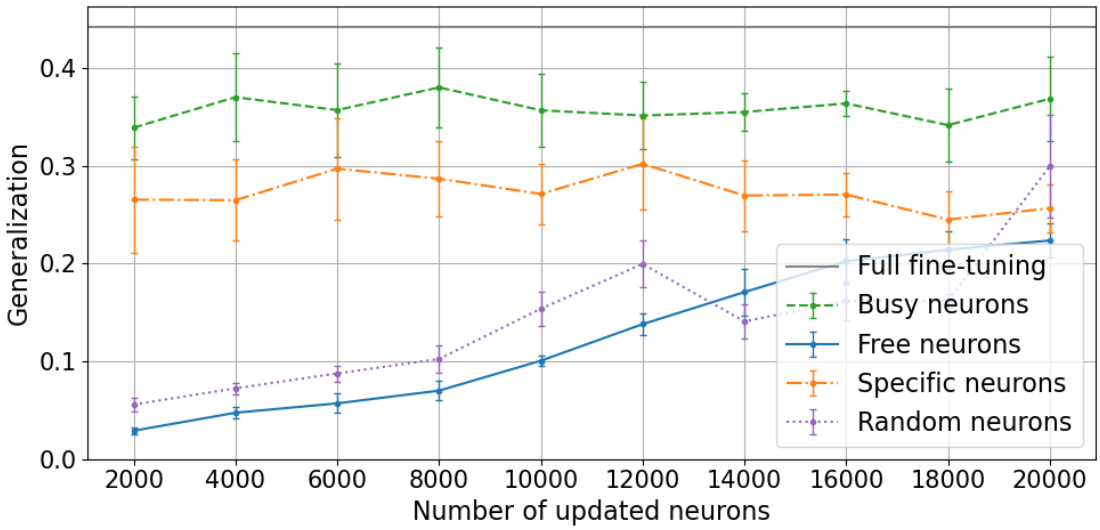

Figure 5.17: Comparison between the generalization score of different techniques when editing 1000 facts.

Description

5.3.1 Incremental editing with sparsity by design

To evaluate the impact of true knowledge disjointness in model editing, we build upon the configuration used in Experiment 5.3. In this approach, after integrating the second set of knowledge into its respective disjoint set, we execute an evaluation pipeline to assess which facts from the initial knowledge remain unaffected. These intact facts serve as a measure of locality, guiding us to perform edits on the same neurons targeted in the previous update. Our goal is to determine whether a specific, sparsity-driven edit that enforces disjointness can enhance specificity. For comparison, we use the targeted edit on the busy neurons set measured in Experiment 5.3.

Results

As expected, since the set of facts to be edited is situated in a free configuration, the knowledge density is lower compared to the targeted update using busy neurons. Consequently, more neurons are required to achieve near-perfect reliability (Figure 5.18). In Figure 5.19, we observe the results related to locality: in this case as well, enforcing sparsity does not significantly impact training performance. The accuracy is comparable to that observed in Experiment 5.3 with free neurons: using a small number of neurons yields good results, but as the size of the targeted set increases, the locality drops to levels similar to those seen with busy neurons. In general, to achieve a reliability comparable to that of busy neurons, forced sparsity delivers similar results in terms of locality with no clear advantages.

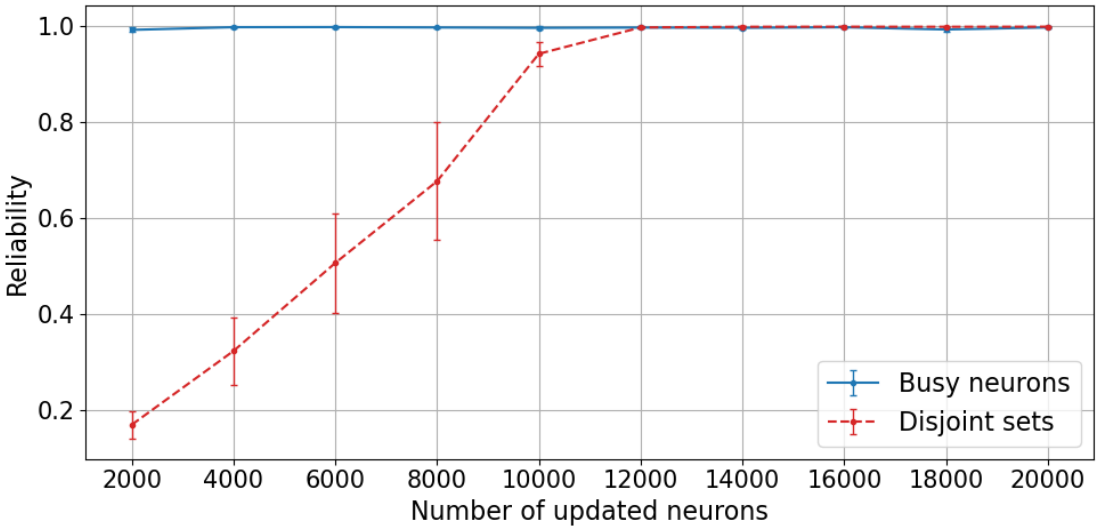

Figure 5.18: Comparison between the reliability score of busy neurons and forced sparsity when editing 1000 facts.

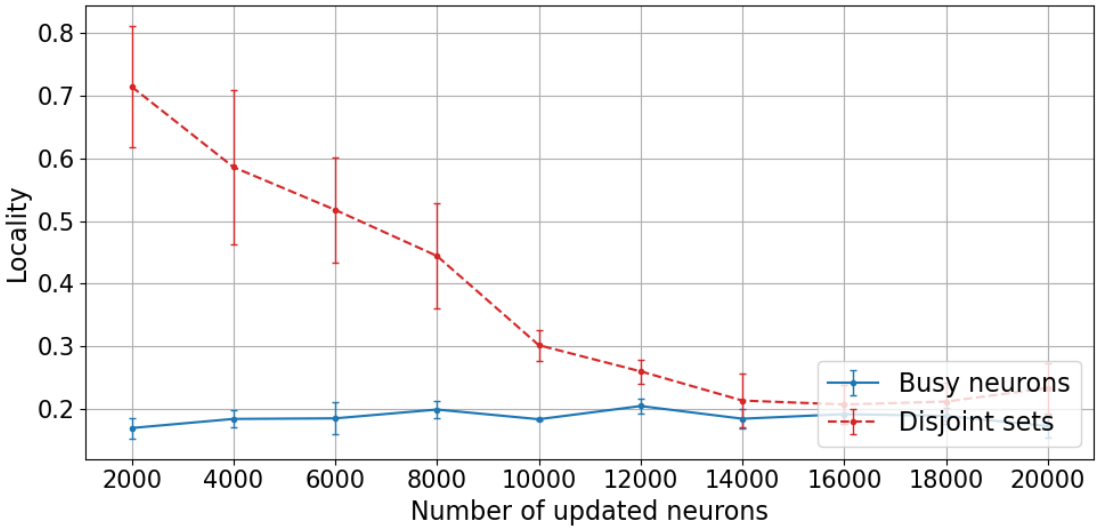

Figure 5.19: Comparison between the locality score of busy neurons and forced sparsity when editing 1000 facts.

Chapter 6

Conclusion

In this chapter, we discuss the final results obtained during the project, highlighting the key findings and their implications. We also present the current limitations and suggest future developments to enhance the outcomes and address any remaining challenges.

6.1 Discussion

In general, the proposed pipeline has been validated through the systematic experiments, providing encouraging results. The familiarity and dissonant detection method proved to have high accuracy across all the classes, in different training scenarios. This method opens different possibilities for future applications not only in the context of cognitive aware learning, but also for detecting possible wrong facts during the training process. The update experiments showed good results in keeping old knowledge while learning new facts when targeting free neurons. With the related ablation studies we proved the analogy between our method and the lottery ticket theory, identifying a set of general knowledge neurons. As for the edit instead, we were not able to reach the same performances shown in the insertion of new knowledge. This is in line with our original theory regarding stubbornness: similarly to what happens in our brain, modifying prior belief is harder than learning new things. Despite the lower results compared to the update, our method outperforms the current state-of-the-art represented by ROME when scaling to large amount of facts. Furthermore, localizing the specific knowledge to be modified is fundamental for the editing process since, inserting dissonant knowledge into the free space creates inconsistencies causing low performance. This core difference between the update and the edit underlines once more the importance of the classification pipeline we built.

Finally, we demonstrated that (1) Inner State Awareness is a valid method for detecting the familiarity and dissonance of factual knowledge inside LLMs; (2) Differentiated Plasticity for targeted updates is a valid method for incremental learning, which effectively distinguish between neurons that have learned previous knowledge and those that are still able to incorporate new facts.

6.2 Limitations and future work

Given the nature of the project, it was only feasible to test the proposed approach on small models with low computational demands. Therefore, a key direction for future work is to scale this method to larger models to assess its validity in more complex scenarios. We are currently setting up experiments with GPT-2 XL and GPT-J for a future article, which will provide valuable insights into the potential real-world applications of this approach. Once we confirm the effectiveness of targeted updates, future research should focus on the impact of the computational graph. As demonstrated, even if knowledge can be localized within specific groups of neurons, the overall neural connections still exert a significant influence on model behavior. The ultimate goal is to develop a method that, considering the computational graph, accounts for signal propagation throughout the network, minimizing internal interference. Moreover, current benchmarks are inadequate for evaluating the true knowledge capabilities of language models, as they often emphasize hard-coded sentences over general concepts. This leads LLMs to excel primarily in scenarios where they can memorize fixed fact structures, which does not align with the principles of generative evaluation. To address this, some have suggested using an external LLM to evaluate responses based on coherence and alignment with the ground truth, rather than simply matching exact phrases [42]. Further studies should delve deeper into novel techniques, such as generative evaluation methods, and exploit them to achieve a more precise evaluation of the knowledge embedded within LLMs.

Bibliography

[1] Mladjan Jovanovic and Peter Voss. *Towards Incremental Learning in Large Language Models: A Critical Review*. 2024. arXiv: 2404.18311 [cs.LG]. URL: https://arxiv.org/abs/2404.18311 (cit. on pp. iii, 13).

[2] Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. *Editing Large Language Models: Problems, Methods, and Opportunities*. 2023. arXiv: 2305.13172 [cs.CL] (cit. on pp. iii, 14–16, 32, 51).

[3] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. *Locating and Editing Factual Associations in GPT*. 2023. arXiv: 2202.05262 [cs.CL] (cit. on pp. v, 2, 19, 20, 31).

[4] Elif Isbel et al. «Neuroplasticity of selective attention: Research foundations and preliminary evidence for a gene by intervention interaction». In: *PNAS* 114.35 (2017), pp. 9247–9254 (cit. on p. 1).

[5] N. Lavi. «Perceptual load as a necessary condition for selective attention». In: *Journal of Experimental Psychology: Human Perception and Performance* 21.3 (2017), pp. 451–468 (cit. on p. 1).

[6] D. E. Broadbent. *Perception and Communication*. London: Pergamon Press, 1958 (cit. on p. 1).

[7] J. A. Deutsch and D. Deutsch. «Attention: Some Theoretical Considerations». In: *Psychological Review* 70.1 (1963), pp. 80–90. DOI: 10.1037/h0042712 (cit. on p. 1).

[8] A. M. Treisman. «The Effect of Irrelevant Material on the Efficiency of Selective Listening». In: *The American Journal of Psychology* 77.4 (1964), pp. 533–546 (cit. on p. 1).

[9] M. Corbetta and G. L. Shulman. «Control of Goal-Directed and Stimulus-Driven Attention in the Brain». In: *Nature Reviews Neuroscience* 3.3 (2002), pp. 201–215. DOI: 10.1038/nrn755 (cit. on p. 1).

[10] M. I. Posner and C. R. R. Snyder. «Attention and Cognitive Control». In: *Cognitive psychology: Key readings* (2004), pp. 205–223 (cit. on p. 1).

[11] N. Yeung. «Conflict Monitoring and Cognitive Control». In: *Oxford Handbook of Cognitive Neuroscience*. Oxford: Oxford University Press, 2013, pp. 275–299 (cit. on p. 1).

[12] D. Frey, E. D. Johnson, and W. De Neys. «Individual differences in conflict detection during reasoning». In: *Quarterly journal of experimental psychology* 71.5 (2018), pp. 1188–1208 (cit. on p. 1).

[13] Janie Brisson, Walter Schaeken, Henry Markovits, and Wim De Neys. «Conflict detection and logical complexity». In: *Psychologica Belgica* 58.1 (2018), pp. 318–332 (cit. on p. 1).

[14] M. Ostrow et al. «How the human brain creates cognitive maps of related concepts». In: *Nature* 632.8026 (2024), pp. 744–745 (cit. on p. 1).

[15] Christine J. Charvet. «Mapping Human Brain Pathways: Challenges and Opportunities in the Integration of Scales». In: *Brain Behavior and Evolution* 98.4 (2023), pp. 194–209 (cit. on p. 1).

[16] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196 [cs.CL]. URL: https://arxiv.org/abs/2402.06196 (cit. on p. 2).

[17] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. *The Unreasonable Ineffectiveness of the Deeper Layers*. 2024. arXiv: 2403.17887 [cs.CL]. URL: https://arxiv.org/abs/2403.17887 (cit. on p. 2).

[18] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. *Fast Model Editing at Scale*. 2022. arXiv: 2110.11309 [cs.LG] (cit. on pp. 2, 22).

[19] Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. *Mass-Editing Memory in a Transformer*. 2023. arXiv: 2210.07229 [cs.CL] (cit. on p. 2).

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL] (cit. on pp. 6–8).

[21] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. «Improving language understanding by generative pre-training». In: (2018) (cit. on p. 9).

[22] Steve D Yang, Zulfikhar A Ali, and Bryan M Wong. «Fluid-gpt (fast learning to understand and investigate dynamics with a generative pre-trained transformer): Efficient predictions of particle trajectories and erosion». In: *Industrial & Engineering Chemistry Research* 62.37 (2023), pp. 15278–15289 (cit. on p. 10).

[23] Jonathan Frankle and Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2019. arXiv: 1803.03635 [cs.LG]. URL: https://arxiv.org/abs/1803.03635 (cit. on p. 12).

[24] Abhishek Aich. *Elastic Weight Consolidation (EWC): Nuts and Bolts*. 2021. arXiv: 2105.04093 [cs.CV]. URL: https://arxiv.org/abs/2105.04093 (cit. on p. 13).

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML]. URL: https://arxiv.org/abs/1503.02531 (cit. on p. 13).

[26] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. *A Survey on Mixture of Experts*. 2024. arXiv: 2407.06204 [cs.LG]. URL: https://arxiv.org/abs/2407.06204 (cit. on p. 13).

[27] Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. *Memory-assisted prompt editing to improve GPT-3 after deployment*. 2023. arXiv: 2201.06009 [cs.CL] (cit. on p. 16).

[28] Zexuan Zhong, Zhengxuan Wu, Christopher D. Manning, Christopher Potts, and Danqi Chen. *MQuAKE: Assessing Knowledge Editing in Language Models via Multi-Hop Questions*. 2023. arXiv: 2305.14795 [cs.CL] (cit. on p. 16).

[29] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. *Memory-Based Model Editing at Scale*. 2022. arXiv: 2206.06520 [cs.AI] (cit. on p. 16).

[30] Zeyu Huang, Yikang Shen, Xiaofeng Zhang, Jie Zhou, Wenge Rong, and Zhang Xiong. *Transformer-Patcher: One Mistake worth One Neuron*. 2023. arXiv: 2301.09785 [cs.CL] (cit. on p. 17).

[31] Qingxiu Dong, Damai Dai, Yifan Song, Jingjing Xu, Zhifang Sui, and Lei Li. «Calibrating Factual Knowledge in Pretrained Language Models». In: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5937–5947. DOI: 10.18653/v1/2022.findings-emnlp.438. URL: https://aclanthology.org/2022.findings-emnlp.438 (cit. on p. 18).

[32] Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. *Modifying Memories in Transformer Models*. 2020. arXiv: 2012.00363 [cs.CL] (cit. on pp. 18, 19).

[33] Nicola De Cao, Wilker Aziz, and Ivan Titov. *Editing Factual Knowledge in Language Models*. 2021. arXiv: 2104.08164 [cs.CL] (cit. on p. 21).

[34] Jinwen He, Yujia Gong, Kai Chen, Zijin Lin, Chengan Wei, and Yue Zhao. *LLM Factoscope: Uncovering LLMs' Factual Discernment through Inner States Analysis*. 2023. arXiv: 2312.16374 [cs.CL] (cit. on p. 24).

[35] Weihang Su, Changyue Wang, Qingyao Ai, Yiran HU, Zhijing Wu, Yujia Zhou, and Yiqun Liu. *Unsupervised Real-Time Hallucination Detection based on the Internal States of Large Language Models*. 2024. arXiv: 2403.06448 [cs.CL] (cit. on p. 24).

[36] Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. *Scaling and evaluating sparse autoencoders*. 2024. arXiv: 2406.04093 [cs.LG]. URL: https://arxiv.org/abs/2406.04093 (cit. on p. 25).

[37] OpenAI. URL: https://huggingface.co/openai-community/gpt2 (cit. on p. 30).

[38] Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. «Measuring and Improving Consistency in Pretrained Language Models». In: *Transactions of the Association for Computational Linguistics* 9 (Dec. 2021), pp. 1012–1031. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00410. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00410/1975957/tacl_a_00410.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00410 (cit. on p. 31).

[39] Wikidata. URL: https://www.wikidata.org (cit. on p. 31).

[40] Peng Wang et al. *EasyEdit: An Easy-to-use Knowledge Editing Framework for Large Language Models*. 2024. arXiv: 2308.07269 [cs.CL]. URL: https://arxiv.org/abs/2308.07269 (cit. on p. 34).

[41] DeepSpeed. URL: https://www.deepspeed.ai/training/ (cit. on p. 35).

[42] Lianmin Zheng et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL]. URL: https://arxiv.org/abs/2306.05685 (cit. on p. 58).