A Comparative Study of Fine-Tuning Pipelines for Integrating Large Language Models in Multimodal Data Analysis

Cătălin Grîu[1]

Supervisor(s): Kubilay Atasu[1]**, Atahan Akyıldız**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Cătălin Grîu
Final project course: CSE3000 Research Project
Thesis committee: Kubilay Atasu, Atahan Akyıldız, Burcu Ozkan

Abstract

While LLMs are proficient in processing textual information, integrating them with other models presents significant challenges. This study evaluates the effectiveness of various configurations for integrating a large language model (LLM) with models capable of handling multimodal data.

We explore the advantages of using pre-trained LLMs for generating text embeddings and the benefits of fine-tuning LLMs for specific tasks. Our investigation includes various fine-tuning strategies, such as Low-Rank Adaptation (LoRA), prompt tuning, and full fine-tuning, applied to both smaller and larger language models. Additionally, we analyze different training setups, including sequential and cascaded training of LLMs and downstream architectures. Our comparative analysis evaluates the performance and cost-effectiveness of these methods. The findings indicate that while full fine-tuning achieves the best results, LoRA offers a practical balance between computational efficiency and model performance. We also highlight the correlation between increased LLM size and corresponding increases in cost and performance.

1 Introduction

In today's data-driven environment, companies generate and utilize a vast amount of information. This data is often organized in tables containing categorical, numeric, and textual columns. These tables can sometimes be relational, allowing for the creation of graph representations. Given the vast quantity of data, manual processing becomes unfeasible and prone to errors, highlighting the necessity for advanced artificial intelligence capabilities. Therefore, developing AI solutions to accurately analyze such diverse and multimodal data formats and extract meaningful insights is imperative.

With developments in transformers [1], [2], large language models (LLMs) [3], [4], and graph neural networks (GNNs) [5], some data analytics tasks can now be automated. However, integrating textual data with other data types and subsequent models poses distinct challenges [6].

A typical approach to managing datasets containing text, as described in [6], [7], involves a two-stage process: an initial embedding phase and a subsequent supervised learning phase. Initially, data is embedded into a format suitable for analysis. Then, subsequent models such as GNNs are used for tasks like classification or prediction.

Although there is already some work on integrating LLMs and GNNs [6], [7], [8], existing approaches either focus solely on textual datasets without covering multimodal data or neglect certain language model fine-tuning techniques.

This paper investigates the integration of pre-trained large language models (LLMs) with a downstream model, specifically the FT-Transformer [2], for processing multimodal tabular data. We evaluate various training configurations, analyzing the effects of training the LLM and the downstream model together versus separately. To capture the most out of textual data, we compare multiple language model fine-tuning strategies, including Low-Rank Adaptation (LoRA), prompt tuning, and full fine-tuning. This comparative analysis aims to assess the performance and cost-effectiveness of these methods. Our setup is flexible, focusing on method differences, which makes our observations applicable to various downstream models.

The structure of this paper is organized as follows: section 2 begins with background information on the key components of our experiments. Section 3 describes the configurations being tested. The experimental setup is detailed in section 4. The results are presented in section 5. Responsible research practices are discussed in section 6, and the conclusion along with future work is covered in section 7.

2 Background

Understanding the concepts discussed in this section is essential for developing the methodology and deciding which configurations to test. Additionally, it helps in interpreting the results. In subsection 2.1, we introduce key concepts related to LLMs. subsection 2.2 explains how these models can be fine-tuned. The various training pipelines are presented in subsection 2.3. Finally, subsection 2.4 introduces the downstream models that are combined with the LLM.

2.1 Large Language Models

Large Language Models (LLMs) are an essential component of natural language processing (NLP) and are focused on understanding and generating text based on statistical distributions. It's important to note the distinction between LLMs and LMs (language models). He et al. [8] use the terms in a way that implies LMs are smaller models primarily used for text embedding, while LLMs refer to much larger models utilized for text generation and reasoning. However, Jin et al. [6] argue that there is no specific threshold between them and that "Large" in LLM refers to the direction of evolution for language models. We adhere to this latter approach.

The Transformer Architecture

Most modern LLMs utilize the Transformer architecture, which has transformed NLP by effectively capturing long-range dependencies within the text and enabling efficient parallel processing [1]. A key innovation of the Transformer model is the self-attention mechanism, which allows the model to weigh the importance of different words in a sentence when encoding or decoding, thereby capturing context more effectively. The Transformer model consists of an encoder, which processes the input text into a dense representation, and a decoder, which generates the output text from this representation. These can be used independently or together for various NLP tasks such as feature extraction, summarization, classification, and more.

Encoder-Only Models

BERT (Bidirectional Encoder Representations from Transformers) is an encoder-only model introduced by [9]. BERT's primary innovation is its use of bidirectional self-attention, allowing the model to consider both the left and right context when generating embeddings. During training, a percentage of the input tokens are randomly masked, and the model is tasked with predicting these masked tokens based on their context. This training method is called Masked Language Modeling (MLM) objective and is mathematically described as:

$$E_{S\sim D}\left[\sum_{s_i\in S}\log p(s_i|s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_N)\right], \quad (1)$$

where S is a sentence sampled from the corpus D, s_i is the i-th word in the sentence, and N is the length of the sentence [6]. This allows BERT to learn deep bidirectional representations of text. Other masked language models have followed, such as RoBERTa (Robustly Optimized BERT) [4], which optimizes BERT's pre-training methodology by using larger mini-batches and more data.

DistilRoBERTa [10] is a distilled version of the RoBERTa model. It leverages knowledge distillation [11], a process where a smaller model (the student) learns to mimic the behavior of a larger model (the teacher) to achieve similar results with reduced computational resources.

Decoder-Only Models

Decoder-only models, such as Mistral 7B [3], GPT-3 [12] use only the decoder stack from the Transformer architecture. These models are particularly effective for generative tasks, as they predict the next token in a sequence. Decoder-only models are typically trained with an autoregressive objective, where the model predicts the next token in the sequence based on the preceding context [13]. This is formulated as:

$$E_{S\sim D}\left[\sum_{s_i\in S}\log p(s_i|s_1,\ldots,s_{i-1})\right]. \quad (2)$$

where S is a sentence sampled from the corpus D and s_i is the i-th word in the sentence [6].

Embedding Extraction

Both encoder-only and decoder-only models are used to generate text embeddings. Encoder models, such as BERT, utilize either the [CLS] token embedding [9] or mean pooling of token embeddings [14]. Mean pooling involves averaging the hidden states across all tokens in the sequence to produce a fixed-size vector representation. Let $H = [h_1, h_2, \ldots, h_n]$ be the sequence of hidden states from the last layer, where h_i represents the hidden state of the i-th token and n is the sequence length. The mean-pooled embedding $\bar{h}$ is calculated as:

$$\bar{h} = \frac{1}{n}\sum_{i=1}^{n} h_i \quad (3)$$

On the other hand, decoder models use the embedding of the last token in the sequence for text representation. This approach works well because, in autoregressive models, the last token's embedding carries information from the entire input sequence, making it a suitable summary of the sequence [15].

2.2 Parameter-Efficient Fine-Tuning (PEFT)

While LLMs like [3], [9] are trained on vast and diverse datasets, fine-tuning allows these models to perform well on specialized tasks by further training them on a smaller, task-specific dataset.

Full fine-tuning

Full fine-tuning involves initializing the model with pre-trained weights and then updating all these weights based on task-specific data through backpropagation, effectively transferring the knowledge acquired during pre-training to the new task [9]. The number of parameters updated is equal to the total number of parameters of the model, which can be quite slow and sometimes unnecessary.

LoRA

Low-Rank Adaptation (LoRA) [16] is a technique that reduces the number of parameters that need to be updated during fine-tuning by decomposing the weight matrices of the model into low-rank matrices (Figure 1).

Starting with a pre-trained weight matrix $W_0 \in R^{d\times k}$, LoRA introduces an additive change, ΔW, represented by a low-rank decomposition BA, where BA has a rank r, considerably smaller than both the input dimension k and the output dimension d. The initial matrix is updated to $W_0 + \Delta W = W_0 + BA$, with $B \in R^{d\times r}$ and $A \in R^{r\times k}$. During training, W_0 remains unchanged and does not receive gradient updates, while A and B are trainable.

As the number of trainable parameters increases, training LoRA roughly converges to training the original model with no additional inference latency. When deployed in production, it can explicitly compute and store $W = W_0 + BA$, and perform inference as usual.

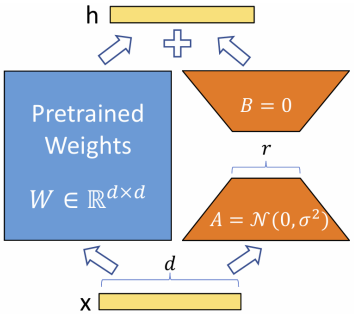

Figure 1: LoRA reparametrization: The left side represents the pre-trained weights of the LLM, while the right side shows the additive change matrix computed during fine-tuning. Only matrices A and B are trainable. Taken from [16].

Prompt Tuning
Prompt tuning [17] is an alternative to traditional fine-tuning that focuses on optimizing the prompts given to a pre-trained language model rather than adjusting its internal weights. In this approach, the model is frozen, and learnable parameters are prepended to the input. These parameters are updated through backpropagation, allowing the adaptation to specific tasks without altering the core structure of the model.

2.3 Training Pipelines

Jin et al. [6] provide a comprehensive survey on combining LLMs and GNNs and categorize the cases based on the role of LLMs and the graph types. For the use of LLMs as encoders, they outline several training pipelines, including one-step training and two-step training.

One-step training involves training the LLM and the downstream model together in a cascaded architecture for downstream tasks. This method integrates textual and structural information simultaneously, allowing the model to learn both representations jointly [18], [19]. While one-step training is convenient, it may suffer from local minima issues where the LLM underfits the data [7].

Two-step training separates the training process into two distinct stages. First, the LLM is fine-tuned on task-specific textual data. Then, the fine-tuned model generates embeddings, which are subsequently fed into a downstream model for further training. In related work, the two-step process is approached differently: some train the LLM only in stage one [7], while others train it in both stages [20].

2.4 FT-Transformer

FT-Transformer is an adaptation of the Transformer architecture designed for tabular data. This model, introduced by Gorishniy et al. [2], transforms both numerical and categorical features into embeddings and processes these embeddings using a stack of Transformer layers. The final representation of the [CLS] token is used for prediction. (Figure 2)

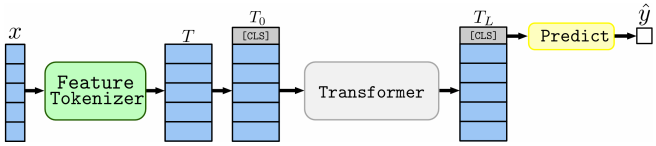

Figure 2: The FT-Transformer architecture. Firstly, Feature Tokenizer transforms features to embeddings. The embeddings are then processed by the Transformer module, with the [CLS] token's final representation used for prediction. Taken from [2].

3 Methodology

In this section, we present the components of the configurations being tested and explain how they interact with each other to illustrate the overall workflow. Our contributions include conducting a series of experiments to evaluate the integration of LLMs with subsequent models that handle tabular data. The study involves constructing a pipeline that starts with embedding textual data using pre-trained or fine-tuned LLMs. These text embeddings are then incorporated into a downstream model, which can be trained alongside or independently of the LLM. The study evaluates and compares these configurations to find the most effective approaches across different datasets and language models.

We begin by explaining how the LLM and chosen downstream model are combined in subsection 3.1. Next, we describe how we train this combined architecture in subsection 3.2. Finally, we outline the fine-tuning techniques applied to the LLM in subsection 3.3.

3.1 Combining LLM with FT-Transformer

Our setup is designed to be flexible, allowing for the selection of a downstream model that can handle multiple modalities in both graph and tabular forms. To test our pipelines, we selected a simple model for handling tabular data: the FT-Transformer [2].

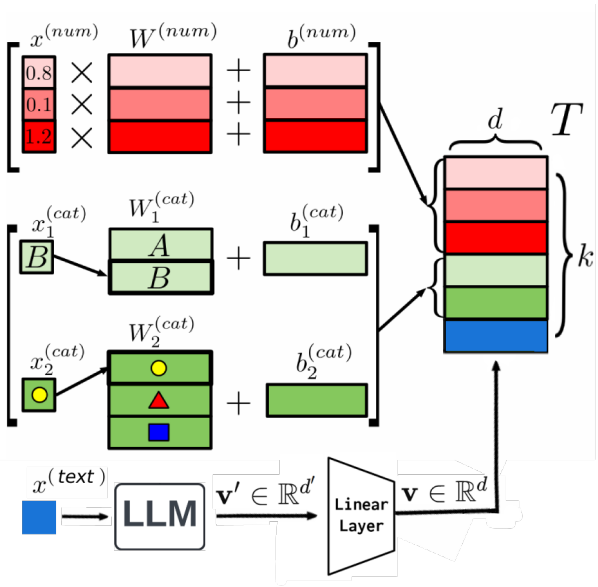

Figure 3: Integration of the LLM with the Feature Tokenizer stage of FT-Transformer. Let $\mathbf{v}'$ denote the embeddings generated by the LLM for text fields, where their size d' depends on the LLM. Using a linear layer, these embeddings are reduced to dimension d, a hyperparameter of FT-Transformer. The resulting matrix $\mathbf{T}$ of concatenated embeddings serves as input to a Transformer architecture. Adapted from [2].

When the LLM is combined with the FT-Transformer, text fields are replaced with their embeddings generated by the LLM. These embeddings are concatenated with embeddings of other modalities created by the Feature Tokenizer part of FT-Transformer. If the embeddings differ in size, a linear transformation is applied to ensure uniform dimension d. Finally, a matrix of dimension d (embedding dimension) by k (number of features) is created for each entry, which is further processed by layers of Transformers. This process is depicted in Figure 3.

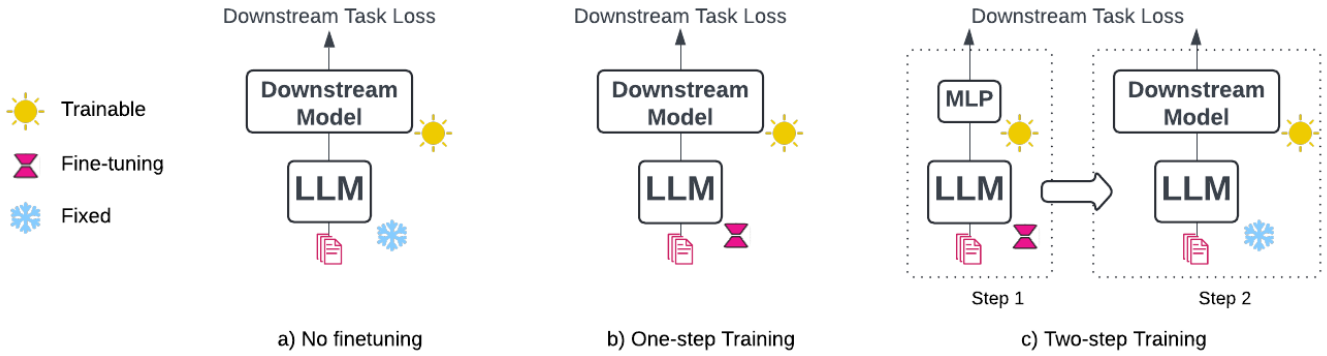

Figure 4: The illustration of 3 training techniques: (a) No Fine-tuning, (b) One-step Training, (c) Two-step Training. Adapted from [6].

3.2 Training Pipelines

We compare three training pipelines depicted in Figure 4. The initial tested pipeline involves **no fine-tuning**: the large language model (LLM) is used solely to generate embeddings, and the downstream model is trained while the LLM remains fixed. It is particularly convenient as it allows for precomputing the text embeddings and reusing them whenever modifications are made to the downstream model.

In the **one-step training** method, after each step, when the weights of the downstream model are updated through backpropagation, the update is propagated further, and the weights of the LLM are updated as well. After each epoch, the new version of the LLM is used to generate embeddings for the text columns.

In the **two-step training** pipeline, stage 1 involves fine-tuning the LLM, which is then used in stage 2 to generate text embeddings without further training. The fine-tuning in stage 1 is performed on the same downstream task as in stage 2, but it utilizes only concatenated text fields. To compute the loss, the extracted embedding (with a fixed size determined by the LLM) is reduced to the desired size using a multi-layer perceptron (MLP) with one hidden layer. The output size is equal to the number of classes for classification or a single unit for regression. Notably, stage 1 by itself is equivalent to the one-step process, with the downstream model being the MLP.

3.3 Fine-tuning Strategies

To fine-tune the LLM on specific datasets, we use several methods: LoRA, prompt tuning, and full fine-tuning presented in subsection 2.2. For LoRA, we conduct a grid search to find the best values for alpha and rank parameters, determining whether a universal set can be applied across all our configurations or if distinct sets are required for each configuration.

In prompt tuning, we need to find the optimal number of virtual tokens added to the input. Adding more tokens increases the input length, which may require more truncation of the original data to fit within LLM's token limit. We experiment with both including and excluding the virtual tokens in the mean pooling process. In the exclusion scenario, the virtual tokens are used solely for influencing the attention mechanisms within the LLM's layers.

4 Experimental Setup

This section begins with a discussion on the chosen datasets in subsection 4.1. Following that, we present the LLMs used for the experiments in subsection 4.2.

4.1 Datasets and evaluation metrics

To increase the flexibility of our study and enable integration with downstream models that handle multiple modalities in both graph and tabular formats, we use datasets that meet these criteria.

One such dataset is the Amazon Fashion dataset, which includes 883,636 reviews of 186,637 products, forming a network that links users and products through reviews. Due to computing constraints, we select a random subset of 100,000 reviews for our experiments. The column names and types, highlighting the multimodal nature of the dataset, are detailed in Table 1. Our task is to predict the correct rating for each review. We use mean squared error (MSE) as our evaluation metric because it effectively measures our model's performance by treating ratings as continuous values and penalizing larger errors more heavily.

Table 1: Overview of Amazon Fashion dataset columns, data types, and descriptions.

Column Name	Data Type	Description
reviewerID	categorical	ID of the reviewer
asin	categorical	ID of the product
overall	numerical	Rating given
reviewText	text	Text of the review
summary	text	Summary of the review
unixReviewTime	timestamp	Time of the review
vote	numerical	Upvotes of the review
verified	categorical	Verification status

The second dataset used in our study is the ogbn-arxiv. It is a citation network consisting of 169,343 nodes (papers)

and 1,166,243 edges (citations). Each node is a paper represented by title, abstract, and year of publication (Table 2). Each directed edge indicates that one paper cites another one. Its extensive use in the research community offers numerous benchmarking opportunities, making it an excellent choice for validating our results. Our task is node classification, for which we use cross-entropy [21] as the loss function and accuracy as the evaluation metric.

Table 2: Overview of ogbn-arxiv dataset columns, data types, and descriptions.

Column Name	Data Type	Description
title	text	Title of the paper
abstract	text	Abstract of the paper
year	numerical	Publishing year
category_index	categorical	Category

4.2 Large Language Model Selection

To select the LLMs for our experiments, we reviewed the state-of-the-art models for text embedding, focusing on sources such as the Massive Text Embedding Benchmark (MTEB) Leaderboard[1] and examples used in related work [22]. Additionally, we considered our computational resource constraints. We chose two models with distinct architectures:

- all-distilroberta-v1[2] - a fine-tuned version of the DistilRoBERTa model presented in subsection 2.1. It has 82.8 million parameters, a context length of 512, and an embedding output size of 768.
- e5-mistral-7b-instruct[3], which ranks among the top 10 models on the MTEB leaderboard. It is a fine-tuned version of Mistral 7b [3], with a decoder-only architecture as described in subsection 2.1. This model has 7 billion parameters, with a context length of 4096, and an embedding size of 4096.

To input our text into the LLM, it must first be tokenized. This process converts the text into smaller units, and maps them to numerical IDs using a fixed vocabulary [23]. We examine the length of the tokenized text to ensure it fits within the LLM's context length (Figure 5). If the tokenized text exceeds the context length, it must be truncated, which can potentially impact the model's performance. For DistilRoBERTa, this occurs in 0.5% of cases with the ogbn-arxiv dataset and 0.08% of cases with the Amazon Fashion dataset. In contrast, for e5-mistral, no entity in either dataset approaches its 4096 context length.

[1] https://huggingface.co/spaces/mteb/leaderboard

[2] https://huggingface.co/sentence-transformers/all-distilroberta-v1

[3] https://huggingface.co/intfloat/e5-mistral-7b-instruct

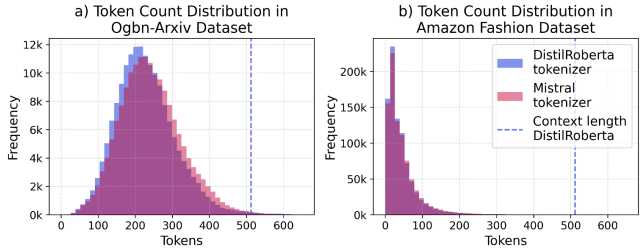

Figure 5: Distribution of token counts using the tokenizers of DistilRoBERTa and e5-Mistral. For each dataset, the text columns were concatenated.

Next, the number of trainable parameters for each LLM and fine-tuning strategy is shown in Table 3. An increase in trainable parameters affects both training time and memory usage.

Table 3: Trainable parameter analysis of DistilRoBERTa and e5-mistral-7b under different fine-tuning strategies.

LLM	LLM Fine-tuning Strategy	LLM Trainable Params
DistilRoBERTa	No fine-tuning	0
	LoRA (rank 64)	1.18M
	Prompt (24 tokens)	18,432
	Full	83.1M
e5-mistral-7b	No fine-tuning	0
	LoRA (rank 64)	27.26M
	Prompt (24 tokens)	98,304
	Full	7.11B

5 Results

The results of our experiments are detailed in Table 4, which compares the configurations made by the two LLMs, across two datasets and several different training pipelines.

Downstream Model

We observe the best results across both datasets and both language models with just the MLP as the downstream model. Further using FT-Transformer for Amazon Fashion to try and take advantage of other modalities did not improve the results. The lack of performance may be attributed to insufficient hyperparameter tuning for the FT-Transformer model. With the ogbn-arxiv, the accuracy of the FT-Transformer is closer to that of the MLP, but it may not have enough modalities and columns to take advantage of.

Fine-tuning pipelines

Our findings indicate that fine-tuning the LLM specifically for each dataset significantly enhances the results. Among the methods tested, full fine-tuning yields the best performance, closely followed by LoRA, and then prompt tuning. Comparing one-step and two-step pipelines for FT-Transformer, the two-step approach consistently yields the best results across all datasets and fine-tuning techniques. This aligns with [7], who reported experiencing convergence issues with the one-step method.

Table 4: Comparison of LLM fine-tuning pipelines and their impact on downstream model performance across multimodal datasets. For each configuration: LLM - Downstream Model - Dataset, we **bold** the best and underline the second-best result.

LLM	Downstream Model	LLM Fine-tuning Pipeline	Amazon Fashion		OGBN-ArXiv	
			MSE ↓	Time	Accuracy ↑	Time
DistilRoBERTa	MLP	No fine-tuning	0.3087	13min	73.62	20min
		One-step (LoRA)	0.2244	5h	**74.42**	5.7h
		One-step (Prompt)	0.3043	4.5h	73.48	5.6h
		One-step (Full)	**0.1972**	5.8h	74.38	6.5h
	FT-Transformer	No fine-tuning	0.5379	30min	73.00	50min
		One-step (LoRA)	0.5196	6.4h	73.22	9.4h
		One-step (Prompt)	0.63	6.4h	73.24	9.4h
		One-step (Full)	0.6012	9.2h	73.22	10.7h
		Two-step (LoRA)	0.5112	5.5h + 30min	73.85	5.7h + 50min
		Two-step (Prompt)	0.5704	4.5h + 30min	73.45	5.6h + 50min
		Two-step (Full)	**0.3598**	5.8h + 30min	**74.02**	6.5h + 50min
e5-mistral-7b	MLP	No fine-tuning	**0.1778**	3h	**76.02**	20h
		One-step (LoRA)	0.1858	60h	75.46	75.7h
	FT-Transformer	No fine-tuning	**0.4544**	10h	**75.61**	20h
		Two-step (LoRA)	0.4857	60h + 3h	75.18	75.7h + 20h

LLM

The difference in size between the models is very observable in our results. The simplest baseline using e5-mistral with no fine-tuning and an MLP manages to outperform all metrics achieved with DistilRoBERTa. Text embedding generation is, however, much more costly. Thus, fine-tuning for the desired number of epochs was unfeasible with our time and computational resources. Our 10 epochs of fine-tuning e5-mistral for Amazon Fashion and 5 epochs for ogbn-arxiv yielded slightly worse results, but the metrics did not converge, and we expect better results with more epochs.

Runtime

The runtime is significantly shorter for pipelines without fine-tuning because the LLM only needs to embed the text once, with the remaining epochs dedicated to training the downstream model. For instance, using the FT-Transformer, the no fine-tuning pipeline for Amazon Fashion takes 30 minutes, which includes embedding the text once and then training the model. In the two-step pipeline, the total time includes both the duration to fine-tune the LLM in stage 1 and the 30 minutes required in stage 2 for text embedding and downstream model training. The runtime of the 3 fine-tuning strategies is proportional to the number of learnable parameters: full fine-tuning consistently takes more time, followed by LoRA and prompt-tuning.

For the e5-mistral model, the runtime is largely dominated by the forward pass. For instance, embedding the ogbn-arxiv dataset took around 20 hours. However, a brief experiment with bfloat16 quantization [24] indicated that the runtime could be reduced by several factors. Given the dominance of the forward pass, the time differences among various fine-tuning techniques are less pronounced. Nonetheless, the choice of technique significantly impacts memory usage, which is proportional to the number of trainable parameters. With our constraints, fully fine-tuning all 7 billion parameters is impractical, as it requires over 100GB of VRAM.

Comparisons

For the ogbn-arxiv dataset, Duan et al. [7] report an accuracy of 74.32 using a 355M parameters language model and an MLP. They further increase the accuracy to 76.18 by using a GNN as downstream model. In comparison, our results demonstrate that with a larger language model, we can achieve a significantly higher accuracy of 76.02 using just an MLP. This accuracy is expected to improve further by incorporating the graph structure of the dataset.

6 Responsible Research

Dataset and Language Model Transparency

In this research, transparency regarding the training data of the language models used is a critical ethical consideration. Selecting models with fully disclosed training data ensures that there is no data leakage when evaluating the model's performance on our chosen datasets. This approach supports the integrity and reproducibility of our research. The training details and data sources for the used LLMs are available on Hugging Face[4].

Following the FAIR principles (Findable, Accessible, Interoperable, and Reusable), we selected well-known public datasets that are easy to locate and have been used in prior research. To ensure a fair comparison with existing benchmarks for the ogbn-arxiv dataset, we followed the default train/validation/test split provided by the ogbn library. This consistency with previous studies enhances the

[4] https://huggingface.co

reproducibility of our methods, enabling other researchers to verify and build upon our work effectively.

Objective Metric Reporting
All experiments using DistilRoBERTa were run for 20 epochs. The "no fine-tuning" pipelines with e5-mistral were also run for 20 epochs. Due to computational constraints, we were unable to fine-tune e5-mistral for more than 10 epochs for the Amazon fashion dataset and 5 epochs for the ogbn-arxiv dataset. We report the best metric achieved for each run. Additionally, for time metrics, we report the fastest time among the same runs.

Hyperparameter Tuning
We conducted hyperparameter tuning using a grid search. Our findings indicated that for LoRA, a rank of 64 and an alpha of 16 worked best across all setups. For prompt tuning, we tested different numbers of virtual tokens, with the best performance observed at 24 virtual tokens, without taking them into the mean pooling process.

Reproducibility
All experiments were conducted on an A6000 GPU. The logs of our results along with the code to reproduce them can be found at https://gitlab.ewi.tudelft.nl/takyildiz/cse3000.

Environmental Impact
One significant advantage of task-specific fine-tuning is its contribution to reducing the overall carbon footprint of AI model training. By leveraging pre-trained models, the need for training models from scratch on large datasets is eliminated, leading to substantial energy savings. Techniques like LoRA and prompt tuning enable efficient fine-tuning by updating only a fraction of the model's parameters, which further reduces computational requirements and energy consumption.

7 Conclusions and Future Work

This research investigates the integration of pre-trained large language models (LLMs) with subsequent models for multimodal data analysis. Various fine-tuning techniques were evaluated across different training setups. The findings provide a valuable guide for making informed decisions in future multimodal data analysis tasks.

We present the performance and trade-offs between different configurations, highlighting that the choice of LLM significantly impacts the results, with larger models like e5-mistral-7b outperforming smaller ones like DistilRoBERTa. Our results indicate that fine-tuning the LLM significantly improves metrics, with both full fine-tuning and LoRa being effective options. Additionally, we find that decoupling the fine-tuning of the LLM from the training of the downstream model is the most effective approach.

Future work could benefit from benchmarking other downstream models, such as combined GNN/tabular transformer architectures, using insights from this study. Another promising direction is to apply our findings to improve self-supervised learning by using masked fields from the dataset as labels. Additionally, future research could explore quantization, specifically using smaller data types like bfloat16, in larger LLMs to evaluate the speed and memory advantages and the associated trade-offs.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 932–18 943, 2021.

[3] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[4] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[5] B. Egressy, L. Von Niederhäusern, J. Blanuša, E. Altman, R. Wattenhofer, and K. Atasu, "Provably powerful graph neural networks for directed multigraphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 11 838–11 846.

[6] B. Jin, G. Liu, C. Han, M. Jiang, H. Ji, and J. Han, "Large language models on graphs: A comprehensive survey," *arXiv preprint arXiv:2312.02783*, 2023.

[7] K. Duan, Q. Liu, T.-S. Chua, *et al.*, "Simteg: A frustratingly simple approach improves textual graph learning," *arXiv preprint arXiv:2308.02565*, 2023.

[8] X. He *et al.*, "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning," *Journal of Machine Learning*, 2024.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[12] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[13] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[14] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, 2019.

[16] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[17] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[18] B. Xin, Y. Zhou, L. Wu, and J. Luo, "Xr-transformers: Cross-resolution transformers for efficient image super-resolution," *arXiv preprint arXiv:2206.04517*, 2022.

[19] C. Author and D. Author, "Adsgnn: Aggregated graph neural networks for edge-level information," *Journal of Advertising Research*, 2023.

[20] V. N. Ioannidis, X. Song, D. Zheng, *et al.*, "Efficient and effective training of language and graph neural network models," *arXiv preprint arXiv:2206.10781*, 2022.

[21] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.

[22] W. Hu, Y. Yuan, Z. Zhang, *et al.*, "Pytorch frame: A modular framework for multi-modal tabular learning," *arXiv preprint arXiv:2404.00776*, 2024.

[23] C. W. Schmidt, V. Reddy, H. Zhang, *et al.*, "Tokenization is more than compression," *arXiv preprint arXiv:2402.18376*, 2024.

[24] D. Kalamkar, D. Mudigere, N. Mellempudi, *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

A Downstream Models

Table 5: Comparison of model parameters for MLP and FT-Transformer across ogbn-arxiv and Amazon Fashion datasets. The model size is influenced by the number of output classes and, for the FT-Transformer, also by the number of categorical values in the datasets.

Downstream Model	Dataset	Model size (Params)
MLP	ogbn-arxiv	207K
	Amazon Fashion	197K
FT-Transformer	ogbn-arxiv	801064
	Amazon Fashion	22M