# INSTRUCTION-BASED FINE-TUNING OF OPEN-SOURCE LLMS FOR PREDICTING CUSTOMER PURCHASE BEHAVIORS

by
HALIL İBRAHIM ERGÜL

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
Nov 2024

# ABSTRACT

INSTRUCTION-BASED FINETUNING OF OPEN-SOURCE LLMS FOR PREDICTING CUSTOMER PURCHASE BEHAVIORS

HALIL IBRAHIM ERGÜL

Data Science M.S.c THESIS, NOV 2024

Thesis Supervisor: Prof. Selim Saffet Balcısoy

Keywords: Large Language Models, Instruction Tuning, LoRA, Deep Learning

In this study, the performance of various predictive models, including probabilistic baseline, CNN, LSTM, and fine-tuned LLMs, in forecasting merchant categories from financial transaction data have been evaluated. Utilizing datasets from Bank A for training and Bank B for testing, the superior predictive capabilities of the fine-tuned Mistral Instruct model, which was trained using customer data converted into natural language format have been demonstrated. The methodology of this study involves instruction fine-tuning Mistral via LoRA (Low-Rank Adaptation of Large Language Models) to adapt its vast pre-trained knowledge to the specific domain of financial transactions. The Mistral model significantly outperforms traditional sequential models, achieving higher F1 scores in the three key merchant categories of bank transaction data—grocery, clothing, and gas stations— that is crucial for targeted marketing campaigns. This performance is attributed to the model's enhanced semantic understanding and adaptability which enables it to better manage minority classes and predict transaction categories with greater accuracy. These findings highlight the potential of LLMs in predicting human behavior and revolutionizing financial decision-making processes

# ÖZET

AÇIK KAYNAKLI LLM'LERIN MÜŞTERI SATIN ALMA DAVRANIŞLARINI TAHMIN ETMEK İÇIN TALIMAT BAZLI İNCE AYARI

HALIL İBRAHIM ERGÜL

VERİ BİLİMİ YÜKSEK LİSANS TEZİ, KASIM 2024

Tez Danışmanı: Prof. Selim Saffet Balcısoy

Anahtar Kelimeler: Büyük Dil Modelleri, Talimat Tabanlı Eğitim, LoRA, Derin Öğrenme

Bu çalışmada, finansal işlem verilerinden tüccar kategorilerini tahmin etmede olasılıklı temel modeller, CNN, LSTM ve ince ayar yapılmış büyük dil modelleri (LLM'ler) dahil olmak üzere çeşitli tahmin modellerinin performansı değerlendirilmiştir. Banka A'dan alınan veri setleri eğitim için, Banka B'den alınan veri setleri ise test için kullanılarak, müşteri verilerinin doğal dil formatına dönüştürülerek eğitildiği ince ayar yapılmış Mistral Instruct modelinin üstün tahmin yetenekleri ortaya konulmuştur. Bu çalışmanın metodolojisi, geniş önceden eğitilmiş bilgi birikimini finansal işlemler alanına uyarlamak için Mistral'ı LoRA (Büyük Dil Modellerinin Düşük Dereceli Adaptasyon Uyarlaması) aracılığıyla talimat ince ayarı yapmayı içermektedir. Mistral modeli, geleneksel sıralı modellerin çok ötesine geçerek, banka işlem verilerindeki üç önemli tüccar kategorisinde—market, giyim ve benzin istasyonları—daha yüksek F1 puanları elde etmiştir, ki bu, hedefli pazarlama kampanyaları için kritik öneme sahiptir. Bu performans, modelin geliştirilmiş semantik yeteneği ve uyarlanabilirliğine bağlanmakta olup, azınlıkta olan kategorileri daha iyi yönetmesini ve daha yüksek doğrulukla tahmin etmesini sağlamaktadır. Bu bulgular, LLM'lerin insan davranışını tahmin etme potansiyelini ve finansal karar alma süreçlerinde devrim yaratma olasılığını vurgulamaktadır.

## ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis supervisor, Professor Selim Saffet Balcısoy, for his invaluable guidance, encouragement, and unwavering support throughout this journey. His insightful advice and mentorship have been instrumental in shaping this work. I also extend my heartfelt thanks to Professor Dr. Burçin Bozkaya for his valuable feedback and constructive criticism, which greatly enhanced the quality of my research.

I am profoundly grateful to UPILY company for their generous support in providing access to their GPU machines, which were essential for the computational aspect of this research.

I extend my sincere appreciation to my dear friend Ahmet Yasin Aytar for his unwavering encouragement and friendship, which kept me motivated throughout this challenging endeavor.

To my family members—Adem Ergül, Halime Ergül, Ayşe Çetinkaya, Şeyma Yiğit, Abdurrahman Çetinkaya, Yusuf Yiğit, and Cemile Esra—I owe endless thanks for their unwavering love, patience, and belief in me throughout every stage of this journey. Without their support, this work would not have been possible. To Cemile Esra, a wonderful new addition to our family who entered my life during the thesis process and quickly became one of its most cherished parts, I offer a special acknowledgment. Her presence has been an incredible gift, bringing joy, peace, and a unique sense of support that I will forever hold dear.

*To My Beloved Mother and Father*

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# 1. INTRODUCTION

Large Language Models (LLMs) have demonstrated extraordinary proficiency in generating text that closely mimics human language, as well as excelling in a wide array of tasks, such as Natural Language Processing, Information Retrieval, and recommendation [8] [9] [10] [11]. More importantly, these large models have also proven to be successful in improving techniques commonly used to study and model human behavior [12]. Extensive research has highlighted their ability to encode rich knowledge and exhibit compositional generalization, which allows them to apply learned concepts to novel scenarios effectively. When given appropriate instructions, these models can leverage their vast knowledge bases to solve previously unseen tasks and achieve remarkable levels of performance [13] [14]. The versatility and depth of understanding that LLMs offer make them particularly well-suited to addressing complex challenges that require not only strong generalization capabilities but also an in-depth grasp of contextual and semantic intricacies. These capabilities position LLMs as transformative tools with the potential to significantly advance various domains, especially for human behavior modeling problems that demand both extensive knowledge and adaptable reasoning.

One such domain that potentially stands to benefit greatly from the advanced capabilities of LLMs is financial prediction. In the context of financial transactions, predicting user preferences based on historical data is a complex sequential task. This complexity arises from the diverse and imbalanced nature of transaction categories as well as different characteristics of customers' demographic features, which necessitates models that can accurately capture and interpret subtle patterns in user behavior. Conventional neural network models that are used for this next category prediction task, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, have been employed to deal with this problem with varying degrees of success [15]. Despite significant advancements in the field, the domain of sequential prediction of human behavior, particularly regarding next merchant category prediction using transactional customer data, remains relatively unexplored. These sequential models, while effective in certain contexts, often struggle with capturing the patterns and semantic nuances inherent in transaction data, particularly when dealing with real-world imbalanced datasets where

minority classes are underrepresented.

This study utilizes the broader capability of LLMs to predict human behavior by applying this to the specific context of financial transactions, where accurate predictions are particularly valuable. By addressing the complex problem of predicting customer purchase behaviors, it has been demonstrated that LLMs solve this issue effectively and with greater efficiency and less modeling effort compared to deep learning models. This approach not only highlights the efficacy of LLMs in understanding and predicting human behavior but also emphasizes their applicability in critical financial contexts. Specifically, the aim of this study is to come up with a novel methodology for predicting the next purchase merchant categories of customers by fine-tuning an open-source LLM and comparing its performance with deep-learning based sequential models. To the best of our knowledge, this is the first study to employ a similar methodology for this specific prediction task with the purpose of human behavior modeling, which distinguishes this work from existing literature. By using datasets from two different banks (Bank A and Bank B), models have been trained on data from Bank A and test their generalization capabilities on previously unseen data from Bank B. The experimental evaluation is designed to illustrate the performance of these models across different sequence lengths of user preferences within a multi-class classification framework.

The training phase utilizes data from 8,154 unique customers of Bank A, comprising prediction instructions and task outputs with the next category as the ground truth. Fine-tuning samples are derived from this dataset to further fine-tune the Mistral 7B Instruct model. In the testing phase, the model is validated using data from 1,000 randomly chosen unique customers from Bank B. This data, paired with prediction instructions, is used to test the inference capabilities of the fine-tuned model, which then outputs predictions for the next merchant category. The pipeline underscores the importance of fine-tuning and cross-validation across different datasets to ensure the model's robustness and predictive accuracy in varied contexts. Notably, the dataset from Bank A spans a distinct time period compared to the dataset from Bank B, which ensures temporal independence between the training and testing phases.

This study addresses several critical research questions that have yet to be thoroughly explored. Firstly, it has been investigated whether a pre-trained LLM can be effectively fine-tuned on transactional tabular data reformatted into personalized instructions for merchant category prediction tasks. Secondly, it has been evaluated the performance of the fine-tuned model against traditional deep-learning-based sequential prediction models. Lastly, the model's performance across various sequential lengths that were not originally part of the training data has been assessed. By answering these questions, this research aims to provide comprehensive insights into the capabilities and limitations of

fine-tuned LLMs in the context of financial transaction prediction.

The contributions of this thesis are the following:

- A novel methodology for predicting customers' next purchase merchant categories by fine-tuning an open-source Large Language Model (LLM), specifically the Mistral Instruct 7B model, on financial transaction data reformatted into natural language instructions. By converting transactional and demographic data into a format suitable for LLMs and applying instruction fine-tuning via LoRA, this research utilizes the enhanced semantic understanding and adaptability of LLMs to capture complex patterns in customer behavior. This approach enables the model to predict transaction categories with greater accuracy, particularly in handling minority classes, which traditional sequential models like CNNs and LSTMs struggle to represent effectively.
- A comprehensive comparative analysis between the fine-tuned LLM and traditional deep-learning models across various sequence lengths and merchant categories, utilizing datasets from two different banks to assess generalizability. The fine-tuned Mistral model significantly outperforms baseline models and achieves higher F1 scores in key merchant categories such as grocery, clothing, and gas stations. These findings highlight the potential of LLMs to revolutionize predictive tasks in finance by offering enhanced adaptability and accuracy which paves the way for future applications of LLMs in modeling complex human behaviors in other domains.

# 2. Related Works

The integration of Large Language Models (LLMs) into behavioral modeling systems signifies a substantial shift in understanding and predicting user preferences. Through using sophisticated linguistic and contextual capabilities of models such as GPT, PaLM, and LLaMA, researchers are applying these models directly to the field of recommendation systems [1]. This section discusses the existing field of sequential human behavior modeling in four parts. The first part focuses on the taxonomy of LLMs and paradigms of model tuning. The second provides an overview of fine-tuning and adapting generative language models for recommendations. The third analyzes studies that utilize language models in both cross-domain and domain-specific tasks in recommendation and behavioral modeling. Finally, the fourth part focuses on sequential models in behavior modeling via deep learning.

## 2.1 Taxonomy of LLMs and Paradigms of Model Tuning

The application of LLMs in recommendation systems can be broadly categorized into three distinct modeling paradigms [1]. As shown in Figure 2.1, the first paradigm is the *LLM Embeddings and Recommendation Systems (RS)* approach, where LLMs act as feature extractors. In this method, user and item features are fed into the LLMs to obtain embeddings that can be utilized across various recommendation tasks. The second paradigm, *LLM Tokens and RS*, extends the first by generating tokens that reflect potential user preferences through semantic analysis, thereby influencing the recommendation process. The third paradigm, which this study follows, is *LLM as RS*. This approach leverages the entire LLM architecture to function as a standalone modeling system, processing profiles, behavior prompts, and task instructions to output viable recommendations.

LLMs in recommendation systems also diverge based on their tuning strategies: the *non-tuning* and *tuning* paradigms. The non-tuning paradigm exploits the inherent zero-shot

or few-shot capabilities of LLMs, where specific prompts are designed to elicit recommendation capabilities without further parameter adjustments. This is often segmented into prompting techniques and in-context learning, both aimed at utilizing the pre-trained model's existing knowledge base without additional training.

In contrast, the tuning paradigm, which this study follows, involves fine-tuning pre-trained LLMs on domain-specific datasets such as transaction or recommendation data. This approach not only adapts the LLM to specific recommendation tasks but also enhances its ability to understand and predict based on user-item interactions, textual descriptions, and contextual data relevant to the users and items. The fine-tuning process modifies the model's parameters to better align with the recommendation task's unique requirements, which may differ significantly from the objectives during the pre-training phase.

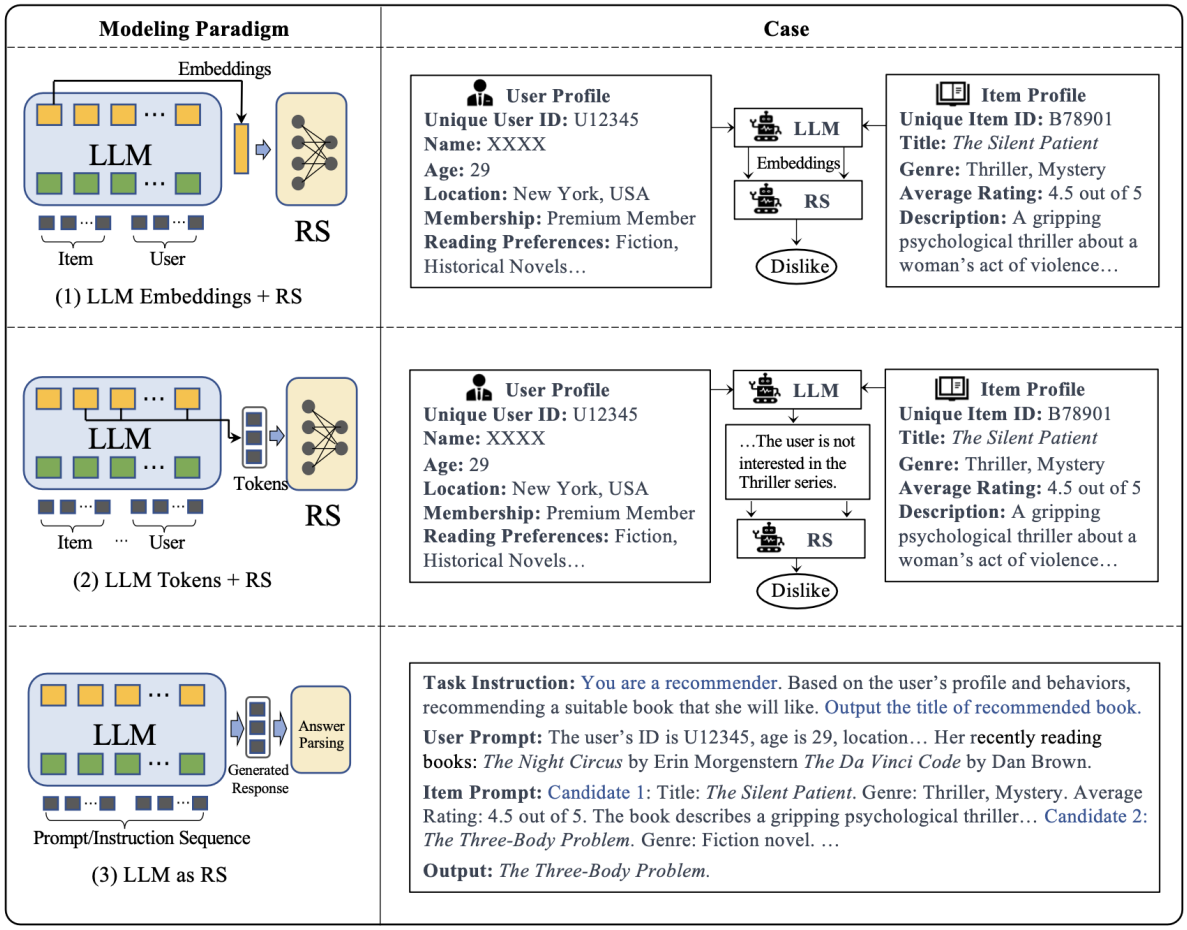


Figure 2.1 An overview of three modeling paradigms [1]

## 2.2 Fine-Tuning and Adapting Generative LLMs for Recommendations

Focusing specifically on generative models within the tuning paradigm, a notable subset is known as *Generative LLM4REC*. This involves models like GPTRec [16], which, unlike BERT4Rec [17] that operates on a discriminative basis, employs a generative approach using techniques like SVD Tokenization for efficiency and adopts a Next-K generation strategy for output generation. This model architecture allows for a more flexible adaptation to the dynamic needs of recommendation systems, particularly in scenarios that involve complex user preferences or varied item categories.

Several innovative models have demonstrated varying degrees of success across different scenarios. For example, the FLAN-T5-XXL model has shown promise in zero-shot, few-shot, and fine-tuned settings, indicating the potential of large-scale fine-tuning in enhancing model performance across different parameter scales [18]. Another model, TALLRec [5], undergoes two stages of tuning: initially using self-instruct data for primary fine-tuning, followed by a recommendation-specific fine-tuning phase that further refines its predictive accuracy based on user feedback.

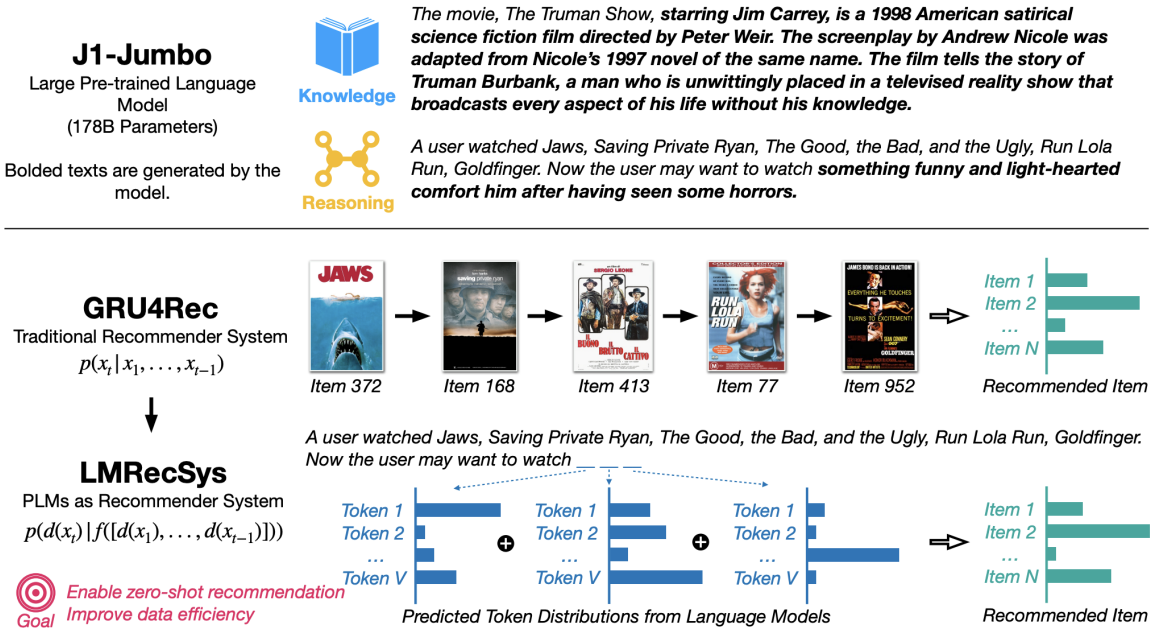


Figure 2.2 Large pre-trained language models are well-suited for recommender systems due to their ability to generate item-specific content (like movie synopses from titles) and infer user preferences from contextual clues. While traditional sequential recommenders operate at the item level, their approach reformulates the recommendation process as a multi-token cloze task using prompts, working at the token level to enable zero-shot recommendations and improve data efficiency. [2]

Further extending the application of LLMs in recommendation systems, Cui et al. introduced M6-Rec, a model that utilizes Alibaba's pretrained LLM, M6 [19]. This model incorporates a minimal addition of task-specific parameters (approximately 1%) to the existing M6 architecture, enabling it to perform a variety of recommendation tasks without altering the foundational model structure. M6-Rec treats recommendation tasks as either language understanding or generation challenges, effectively avoiding the use of user or item IDs, which facilitates its application in open-domain recommendations. This approach highlights the versatility of generative pretrained language models in adapting to diverse domains and tasks which involve unseen items.

Zhang et al. explore the practical implementation and inherent challenges of using LLMs as recommender systems in their study [2]. They specifically address the transformation of session-based recommendation tasks into multi-token cloze tasks using personalized prompts as shown in Figure 2.2. Through MovieLens-1M dataset, the researchers employ a rule-based system to create prompts that guide the LLM, such as GPT-2, to predict the next likely item a user might engage with. Their findings indicate that while the pretrained language model outperforms basic random recommendation baselines in zero-shot settings, it exhibits linguistic biases and underperforms compared to more traditional models like GRU4Rec in few-shot scenarios. This highlights the critical need to balance model training and the design of task-specific adaptations to mitigate biases and enhance performance in real-world applications.

Wang et al. introduced a novel approach where they utilize a prompting strategy to perform next-item recommendations in a zero-shot setting [20]. This method primarily revolves around three operational steps to mitigate the limitations observed in direct querying of LLMs like GPT-3 for recommendations. Initially, they address the broad recommendation space issue by generating a focused candidate set through either user-filtering or item-filtering, which aids in reducing the scope of potential recommendations. Subsequently, they engage GPT-3 in a structured three-step prompting process that begins with summarizing user preferences, followed by selecting representative movies from the candidate set, and culminates in generating a recommendation prompt that guides the model to propose ten closely aligned movies. The final step involves a rule-based extraction method that parses GPT-3's output to obtain the final list of recommended items. This methodology shows an effective integration of external filtering mechanisms and structured LLM prompting to enhance recommendation accuracy in zero-shot scenarios.

Gao et al. explore the enhancement of conversational recommender systems through their study [3] which has a multi staged architecture as shown in 2.3. By incorporating user profiles, historical interactions, and optional dialogue histories into the recommendation

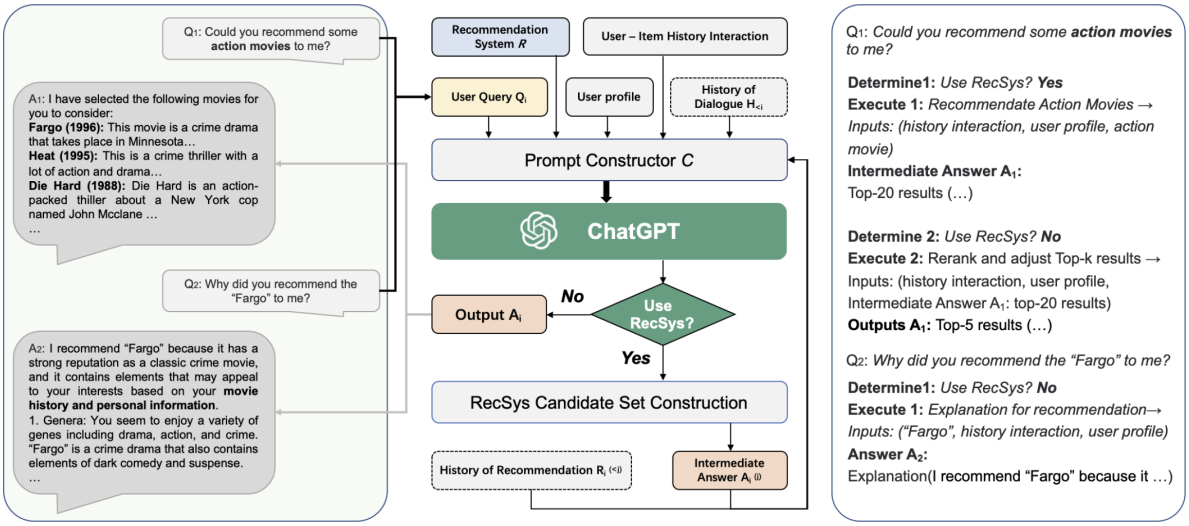


Figure 2.3 Architecture of Conversational Recommendation: ChatREC [3]

process, they utilize these inputs to craft personalized prompts that are then processed by OpenAI's ChatGPT. This system not only summarizes user preferences but also refines a large set of candidate items to a tailored recommendation list. The utilization of ChatGPT enables interactive and multi-round recommendation capabilities which provides a platform for dynamic user interaction and improved explainability of recommendations. Furthermore, they address the challenge of recommending new or less common items by incorporating an item-to-item similarity approach that leverages external, up-to-date information sources. The results from their experiments indicate that ChatREC performs robustly across zero-shot and cross-domain tasks.

Another study introduces a novel approach to news recommendation systems [21]. Conventional methods, which follow the vanilla pre-train and fine-tune paradigm, often fail to fully leverage the rich semantic information that LLMs acquire during pre-training. To address this gap, this paper proposes the Prompt Learning for News Recommendation (Prompt4NR) framework. This framework innovatively applies the pre-train, prompt, and predict paradigm to the news recommendation domain by treating the task as a cloze-style mask-prediction task. This allows for the use of various prompt templates—discrete, continuous, and hybrid—to effectively tailor the model's focus on predicting user clicks on news articles. Additionally, the use of prompt ensembling to aggregate predictions from multiple templates demonstrates a strategic application of LLM capabilities to enhance recommendation accuracy. Extensive experiments on the MIND dataset underscore the effectiveness of Prompt4NR.

Chen and Zhang explore the potential of LLMs to provide explainable recommendations through advanced prompt learning techniques [22]. Traditional methods utilizing recurrent neural networks have not fully capitalized on the capabilities of pre-trained

Transformer models, particularly in integrating user and item IDs, which exist in a different semantic space from the words on which these models were originally trained. This paper introduces two novel strategies—discrete and continuous prompt learning—to integrate these IDs into the recommendation process effectively. In the continuous prompt learning strategy, ID vectors are initially randomly initialized but are refined through sophisticated training strategies such as sequential tuning and recommendation as regularization. These methods significantly improve the model's ability to generate explainable recommendations that are both accurate and user-intelligible.

Hou and Mu shift the focus towards the development of a universal sequence representation learning (SRL) method, named UniSRec, which addresses the limitations of existing SRL methods that rely heavily on explicit item IDs [23]. These methods often struggle to transfer to new recommendation scenarios due to their dependency on item-specific data. UniSRec overcomes this by using item description texts to develop more transferable item representations. This is achieved through a novel item encoding architecture that incorporates parametric whitening and a mixture-of-experts enhanced adaptor. The approach also introduces two contrastive pre-training tasks designed to handle multi-domain negatives. The ability of UniSRec to perform under both inductive and transductive settings, coupled with its strong transferability as evidenced in cross-platform settings, highlights its potential to redefine the adaptability of sequence representation models in recommendation systems.

### 2.3 LLM for Sequential Models in Cross- and Domain-Specific Tasks

In a study by Geng et al., the concept of *Recommendation as Language Processing (RLP)* is proposed, introducing the Pretrain, Personalized Prompt, and Predict Paradigm (P5) [4]. This framework integrates various recommendation tasks such as sequential recommendation, rating prediction, explanation generation, review summarization, and direct recommendation into a unified text-to-text approach as shown in 2.4. The model leverages multitask learning by converting all user and item data into natural language sequences, which are then processed through an encoder-decoder Transformer model. This model is pretrained with instruction-based prompts which allow for effective learning of the semantics necessary for making personalized recommendations. The innovative use of adaptive personalized prompt templates enables the P5 framework to handle conditional text generation tasks efficiently which demonstrates significant improvements over traditional RNN and CNN-based deep learning models in recommendation systems.

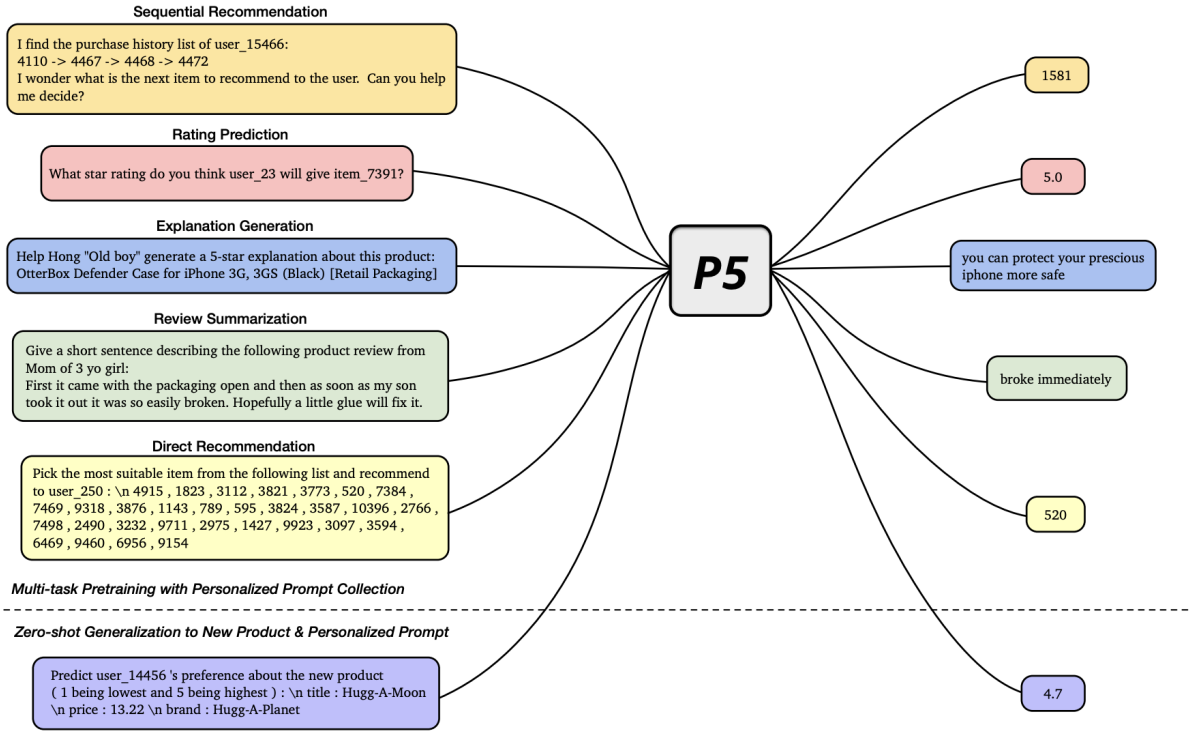


Figure 2.4 An overview of the complete process for different tasks in P5 [4]

Most relevant and similar in terms of training method to the research in this study, Bao and Zhang propose a dual-stage tuning process that significantly augments the recommendation capabilities of LLMs [5]. The TALLRec framework, as shown in 2.5, initiates with an instruction tuning phase that adapts the LLM to understand and process recommendation-specific instructions. This is followed by a recommendation-tuning stage where the model is fine-tuned on actual recommendation data involving user-item interactions in specific domains such as movies and books. Remarkably, even with a limited dataset of fewer than 100 samples, TALLRec demonstrates substantial improvements over traditional models like RNNs and CNNs, achieving scores around 0.70 ($\pm$ 3). Additionally, the model exhibits strong cross-domain generalization capabilities, underscoring its potential in broader application scenarios beyond the initial training domains. This tuning approach aligns closely with the methodology of this study in terms of offering a pertinent example of how structured tuning can enhance the predictive accuracy and domain adaptability of LLMs in recommendation systems.

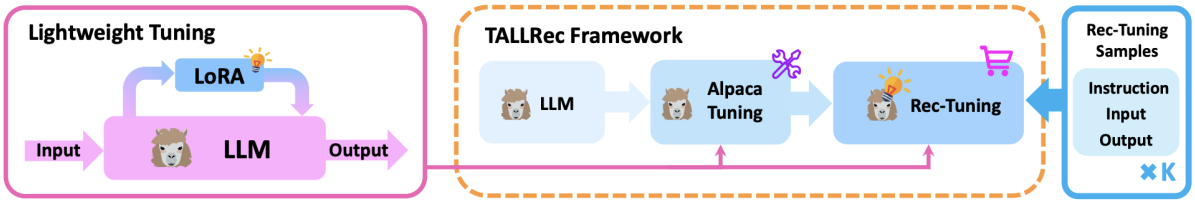


Figure 2.5 TALLRec Dual Modeling Architecture [5]

The novel domain-specific framework PALR illustrates a sophisticated integration of user interaction data with a fine-tuned LLM to enhance item ranking [24]. Unlike conventional approaches that employ smaller LLMs, PALR utilizes a 7-billion-parameter LLM, fine-tuned specifically for ranking, allowing it to better harness the model's reasoning abilities and leverage rich item side information. The system begins with using user/item interactions for candidate retrieval, followed by the LLM-based ranking model, which takes these candidates in a natural language format and uses them to generate personalized item recommendations. Experimental results demonstrate that this approach significantly outperforms existing models on sequential recommendation tasks.

Building on the use of LLMs for recommendation in a cross-domain manner, some studies further explore the capabilities of off-the-shelf pretrained LLMs in a zero-shot learning context [25, 26]. These studies propose frameworks where LLMs are used to estimate the affinity between users and items based purely on textual prompts derived from user preferences, such as movies liked by the user. By formulating the recommendation problem as a conditional ranking task, these frameworks utilize the general-purpose task-solving abilities of LLMs, such as GPT-4, to perform without prior specific training on recommendation data. Despite challenges like the models' struggles to perceive the order of historical interactions and biases towards popular items, carefully designed prompting and bootstrapping strategies significantly mitigate these issues. This enables LLMs to effectively rank items in a zero-shot manner.

Additionally, Ding and Ma introduce the ZESRec algorithm, which uniquely addresses the challenge of data scarcity in new or early-stage recommendation systems [27]. ZESRec operates in a zero-shot learning setting where there are no overlapping users or items between the training and target datasets. By using items' generic features and sequential user interaction histories, ZESRec effectively generalizes across completely new datasets that show the potential to circumvent the typical cold-start problem faced by startups in accumulating sufficient user interaction data to train effective recommendation models. It features four distinct characteristics: cold users (no shared users between training and testing datasets), cold items (no shared items between training and testing datasets), a domain gap (training and testing datasets come from different domains), and no access to target data during training. ZESRec significantly exceeds the performance of zero-shot embedding-KNN and random baselines, offering substantial benefits for startups and early-stage products with limited data.

Figure 2.6 shows the overall framework for item representation and training for VQ-Rec, introduced by Hou and He. It presents an innovative approach for enhancing the transferability of recommender systems through Vector-Quantized item representations [6]. This method addresses the limitations of previous models that heavily relied on pre-trained

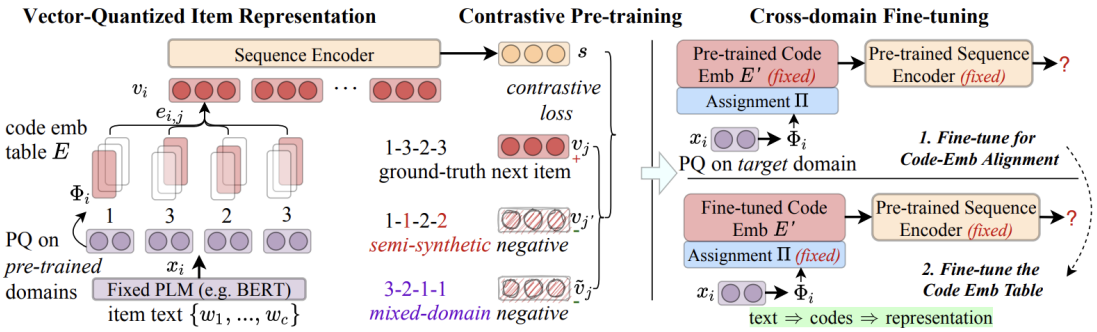


Figure 2.6 VQ-Rec framework for vector-quantized code-based transferable sequential recommender [6]

language models (PLMs) to encode item text into representations, which often resulted in an overemphasis on text features and exaggerated the negative impacts of domain gaps. VQ-Rec innovatively maps item text into discrete indices (item codes), which are then used to retrieve item representations from a code embedding table, following a "text-code-representation" scheme. This novel representation enables the system to mitigate issues tied to domain-specific biases and enhances its adaptability across different domains. The model also incorporates an enhanced contrastive pre-training strategy using semi-synthetic and mixed-domain code representations as hard negatives and a novel cross-domain fine-tuning method employing a differentiable permutation-based network. Extensive testing on six public benchmarks has shown the effectiveness of VQ-Rec, particularly in cross-domain and cross-platform applications. This demonstrates its potential to set a new standard for transferable sequential recommendation systems.

Together, these studies demonstrate the transformative potential of LLMs in recommender systems. This shows how these models can be adapted to tackle both the nuances of personalization and the challenges of zero-shot recommendation. They highlight the increasing feasibility of deploying LLMs in scenarios where traditional methods may falter due to limitations in data availability or the need for extensive retraining. Through the strategic use of natural language understanding and generation capabilities inherent in LLMs, recommender systems can achieve higher accuracy, better user satisfaction, and greater scalability across diverse application domains.

## 2.4 Sequential Models in Behavior Modeling via Deep Learning

The main pipeline of this study resonates with sequential human behavior modeling, focusing on sequences of historical interactions to predict a user's next interaction type or item. While Markov chains played an important role in early sequential recommendation [28, 29, 30], deep learning models have surpassed them in terms of success and ability to model more complex behavior patterns.

The literature on sequential models in recommender systems demonstrates a rich exploration of deep learning (DL) techniques, addressing both theoretical and practical aspects to enhance recommendation accuracy and relevance. Fang and Zhang explore the systematic classification of sequential recommendation tasks based on behavior sequences such as experience-based, transaction-based, and interaction-based [31]. This taxonomy facilitates a deeper understanding of the representative DL-based algorithms tailored for these tasks. The paper emphasizes the crucial factors influencing the performance of DL models in recommender systems, including aspects of model input, data processing, structure, and training. Such comprehensive evaluations illuminate the nuanced impacts these factors have on enhancing recommendation systems.

In the realm of challenges and developments, Wang and Zhang identify and categorize key issues faced by sequential recommender systems (SRSs) [32]. They highlight several challenges, such as managing long or noisy user-item interaction sequences, those with flexible order, heterogeneous relations, and hierarchical structures. This critical analysis of data characteristics within SRSs paves the way for addressing these complexities with innovative deep learning approaches.

Cui and Wu introduce a novel approach to handle the item cold-start problem in network applications [33]. Recognizing the limitations of existing matrix factorization, Markov chain, and RNN methods in capturing dynamic user interests or incorporating additional information, they propose a Multi-View Recurrent Neural Network (MV-RNN). This model integrates visual and textual information through various combinations of multi-view features at the input stage, such as concatenation and fusion techniques. By applying a recurrent structure, MV-RNN dynamically captures user interests and explores different structures in the hidden state to effectively utilize multi-view features. Tested on real-world datasets, MV-RNN demonstrates a significant enhancement in generating personalized rankings and addressing the cold-start problem by accommodating missing modalities.

Donkers et al. extend traditional recurrent neural networks to better suit the specific needs of the recommender systems domain [34]. They emphasize the incorporation of user-specific characteristics into the model to propose a modified type of Gated Recurrent Unit that integrates sequences of consumed items with unique user representations. This approach significantly improves the personalization of recommendations to offer

next-item suggestions that are tailored to individual preferences. The model's effectiveness is validated through offline experiments on two datasets, where it surpasses both conventional RNNs and other state-of-the-art recommendation methods in performance metrics.

Focusing on practical implementations with a newly developed model called GRU4REC, Hidasi and Karatzoglou apply recurrent neural networks (RNNs), specifically using a gated recurrent unit (GRU), to session-based recommendation scenarios where user data is limited [35]. This study introduces modifications such as a ranking loss function and session-parallel mini-batches, which significantly improve the system's ability to provide relevant recommendations based on short session data. This approach underlines the adaptability of RNNs in enhancing session-based recommender systems, demonstrating marked improvements over conventional methods.

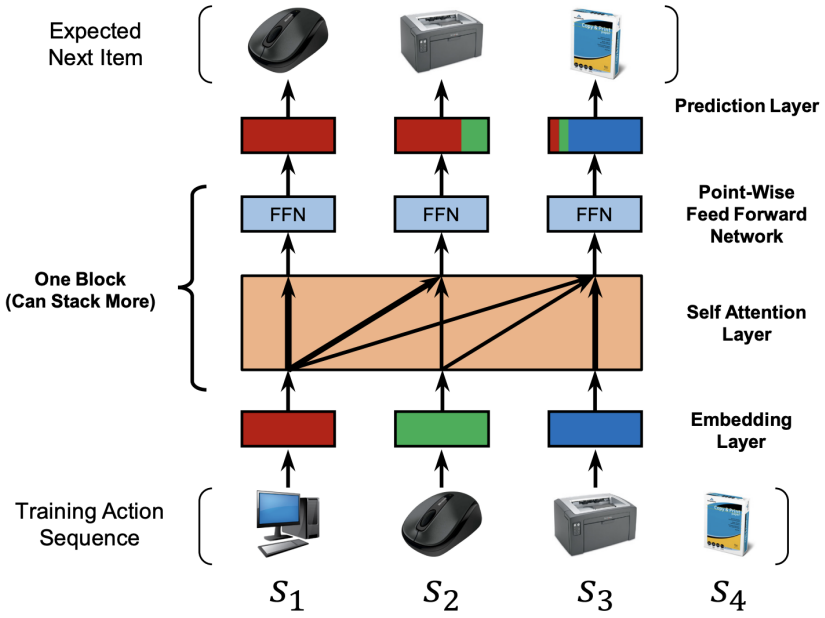


Figure 2.7 SASRec training process. At each time step, the model considers all previous items, and uses attention to 'focus on' items relevant to the next action [7]

Expanding upon the idea of capturing long-term user behaviors, Kang and McAuley propose the SASRec architecture, which employs a self-attention-based sequential model to balance the benefits of both Markov Chains (MCs) and Recurrent Neural Networks (RNNs) [7]. MCs typically predict a user's next action based on a few immediate past actions where the data is sparse. Conversely, RNNs are better suited for denser datasets where uncovering long-term semantics is crucial due to their higher complexity. SASRec innovatively merges these approaches using an attention mechanism that assesses the relevance of past actions at each step, selecting pertinent items to predict subsequent ones as shown in Figure 2.7. This method significantly outperforms various established sequential models across both sparse and dense datasets and showcases greater efficiency

than traditional CNN/RNN-based approaches. The visualization of attention weights in SASRec also offers insights into how the model adapts to different dataset densities and reveals meaningful activity sequence patterns.

Tang and Wang propose a model called Caser, which presents a unique method of treating item sequences as "images" in time and latent spaces [36]. This model uses convolutional filters to learn sequential patterns as local features of these images, providing a unified and flexible network structure that adeptly captures both general user preferences and specific sequential patterns. By embedding recent interactions into a sequence and using convolutional techniques, the Caser model adeptly predicts the top-N items a user is likely to engage with in the near future. Demonstrated on public datasets, the model consistently outperforms other advanced sequential recommendation methods across various evaluation metrics.

Yuan et al. make a significant enhancement in the utilization of Convolutional Neural Networks (CNNs) for session-based next-item recommendations [37]. Traditionally, CNNs embedded an ordered collection of past user interactions into a two-dimensional latent matrix, treating it similarly to an image where convolution and pooling operations are applied. However, this paper identifies the limitations of such models, particularly in handling long-range dependencies in the item sequence. To overcome these, the authors propose a new generative model utilizing a stack of dilated convolutional layers. These layers increase the receptive field efficiently without the need for pooling operations. Additionally, the integration of a residual block structure aids in optimizing deeper networks. This new model not only achieves state-of-the-art accuracy in next-item recommendations but also reduces the training time.

Overall, these studies highlight the evolution of sequential models in behavior modeling via deep learning by emphasizing the shift from traditional methods like Markov chains to more advanced architectures such as RNNs, CNNs, and attention mechanisms. They underscore the importance of capturing complex user behavior patterns and the effectiveness of deep learning techniques in enhancing the predictive capabilities of recommender systems.

# 3. Architecture

## 3.1 Sequantial Prediction Problem

Sequential prediction is a common problem in various fields such as e-commerce, recommendation systems, and financial forecasting. The goal is to predict the next action or event in a sequence based on a user's historical behavior. This type of problem often involves time-ordered data where the sequences of events or interactions evolve over time, making it crucial to capture temporal dependencies to enhance predictive accuracy.

In mathematical terms, the sequential prediction problem can be formulated as follows. Consider a set of users denoted by $U = \{u_1, u_2, \ldots, u_N\}$ and a set of items or events denoted by $V = \{v_1, v_2, \ldots, v_M\}$. Users interact with these items through various behaviors, such as clicking, purchasing, or viewing, which are represented as $B = \{b_1, b_2, \ldots, b_K\}$. Each interaction of a user $u \in U$ with an item $v \in V$ at a specific time $t$ can be associated with a behavior $b \in B$. Thus, the sequential history of user $u$ can be denoted as:

$$S_u = \{(v_1^u, b_1^u, t_1^u), (v_2^u, b_2^u, t_2^u), \ldots, (v_T^u, b_T^u, t_T^u)\}$$

Here, $S_u$ represents the sequence of user $u$'s historical interactions, where $v_i^u$ denotes the $i$-th item interacted with, $b_i^u$ is the behavior type associated with the $i$-th interaction, and $t_i^u$ is the timestamp of the interaction. Given this sequence of historical interactions, the task is to predict what item $v_{T+1}^u$ user $u$ will interact with next, under a specific behavior $b_{T+1}^u$.

The formal task of sequential prediction can be expressed as estimating the probability distribution over future items and behaviors:

$$P(v_{T+1}^u, b_{T+1}^u \mid S_u)$$

where $P(v_{T+1}^u, b_{T+1}^u \mid S_u)$ represents the probability that user $u$ will interact with item $v_{T+1}^u$ with behavior $b_{T+1}^u$ given the historical sequence $S_u$. This formulation captures the essence of sequential prediction, where the goal is to learn a model that can generalize from past sequences to accurately predict future interactions [35, 38].

In real-world applications such as e-commerce or financial transactions, sequential prediction models are employed to forecast what a user will buy or where they will make their next purchase based on past behaviors. For example, in a financial transaction prediction task, the items $V$ can be different merchant categories (such as groceries or gas stations), and the task would be to predict, similar to this study, the next category where a user is likely to make a transaction [5, 39].

Sequential models, particularly Recurrent Neural Networks (RNNs) and their variants like Long Short-Term Memory (LSTM) networks, are commonly employed to capture the temporal dependencies in such prediction tasks [40]. However, these models often struggle with longer sequences and complex behaviors, which has led to the development of more advanced approaches like Transformer-based architectures, that can better capture long-term dependencies and context [41].

The flexibility of sequential prediction models allows them to be adapted to various domains. For instance, in e-commerce, behaviors could include clicking, purchasing, adding to a shopping cart, or adding to favorites. The task then becomes predicting which action a user is likely to take next, given their past interactions. Similarly, in financial contexts, sequential prediction might focus on predicting which merchant category a user will interact with next, based on their transaction history [35, 40].

## 3.2 Research Problem Definition

In this study, the research problem centers around the challenge of accurately predicting the next merchant category in financial transactions using customer behavioral data through a novel method. This task is inherently complex due to several factors: the sequential nature of transactions, the imbalanced distribution of merchant categories, and the varying lengths and patterns of customer purchase histories. Traditional machine learning and deep learning models, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, have been employed in similar tasks, but they often struggle with generalization, especially when faced with long-term dependencies and minority categories. The problem becomes even more pronounced when these models are applied across different datasets from distinct institutions, such as using training data from one bank and testing it on another to measure the generalizability of the model, where discrepancies in transaction patterns and customer demographics add further complexity. This study addresses this gap by proposing the use of fine-tuned open-source Large Language Models (LLMs) to improve the accuracy of next merchant category predictions. Specifically, the research aims to explore whether LLMs, which have demonstrated superior generalization and semantic understanding in other domains, can outperform traditional models in sequential prediction when fine-tuned on transactional data reformatted into personalized instruction sets, and whether these models can effectively handle imbalanced datasets and generalize across different financial institutions and time intervals.

## 3.3 Framework and Pipeline

Figure 3.1 illustrates the main case study of this study that contains the model training and testing pipeline designed for predicting the next merchant category based on historical customer transactions and demographic data. The framework of this study is designed to fine-tune and evaluate the performance of a Large Language Model (LLM) in predicting the next merchant category based on customer transaction histories and demographic data. The pipeline is structured into two primary phases: training and testing. In the training phase, the study utilizes a dataset from Bank A, comprising transaction data from 8,154 unique customers. This data includes the transaction history, demographic

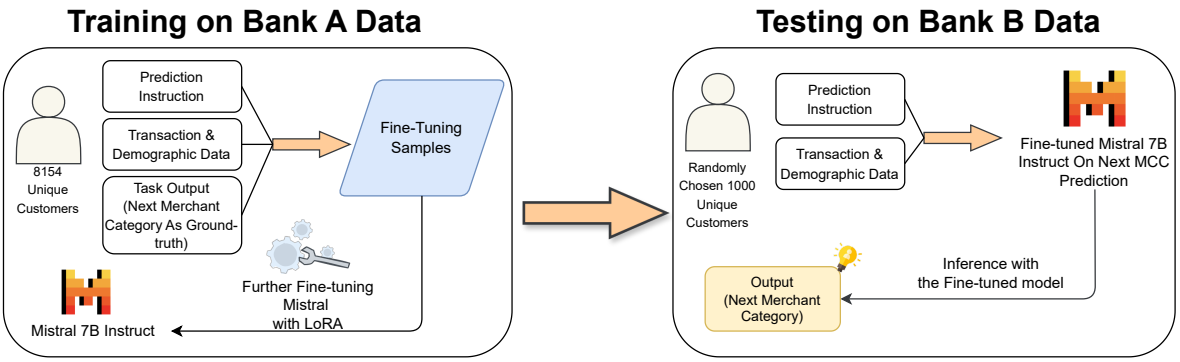


Figure 3.1 An overview of the complete process for fine-tuning and evaluating the Mistral 7B Instruct model on predicting next merchant categories by using distinct datasets from Bank A for training and Bank B for testing to evaluate generalizability.

details, and the merchant category of each transaction, which serves as the ground truth. The raw transactional data is preprocessed and transformed into a natural language instruction format, enabling the Mistral 7B Instruct model to be fine-tuned using this task-specific data. Fine-tuning is achieved via Low-Rank Adaptation (LoRA), allowing the model to adapt its pre-trained parameters to the domain-specific context of financial transactions.

The testing phase involves validating the fine-tuned model on a separate dataset from Bank B, consisting of 1,000 randomly selected customers. This dataset is not used during training to ensure temporal and contextual independence, which helps evaluate the model's generalizability. Similar to the training phase, the Bank B data is reformatted into instruction-based tasks to facilitate inference. The fine-tuned model is tasked with predicting the next merchant category for each customer based on their transaction history and demographic information. Performance is assessed across various sequence lengths to examine the model's ability to predict the next transaction category with varying transaction histories. Cross-validation using distinct datasets (Bank A for training and Bank B for testing) ensures the robustness of the model in predicting merchant categories in diverse contexts.

## 3.4 Open Source LLMs Model Architectures

In this section, the architectures of open-source large language models (LLMs) that were part of the training and inference experiments will be explored, with a focus on their architectures, training methodologies, and performance optimizations. Specifically, the

LLaMA and Mistral models will be examined to highlight key architectural features such as pre-normalization, rotary positional embeddings, and attention mechanisms that enhance efficiency and task-specific capabilities. Additionally, the fine-tuning processes, including supervised techniques and human feedback, will be discussed. Comparative analysis of model performance across various natural language processing tasks will be conducted to provide a comprehensive understanding of the state-of-the-art in open-source LLMs.

## 3.5 LLaMA Models

The LLaMA-2-Chat 7B and LLaMA-3-8B Instruct models, fine-tuned using the Alpaca method [42, 43], are auto-regressive transformer-based language models developed by Meta AI. These models have been optimized through Supervised Fine-Tuning (SFT) and Reinforcement Learning with Human Feedback (RLHF), focusing on enhanced alignment with human preferences for dialogue and task-specific performance.

LLaMA models are trained on a diverse mixture of publicly available datasets like CommonCrawl [42]. These datasets ensure that the models are capable of handling a wide variety of natural language tasks while relying solely on open-source data.

Architecturally, LLaMA employs several enhancements over the standard transformer architecture:

- **Pre-normalization**: RMSNorm is applied to the input of each transformer sub-layer, following the approach in GPT-3, to improve training stability [42].
- **SwiGLU Activation**: ReLU is replaced by SwiGLU, which enhances the model's performance by increasing its capacity for non-linearity [42].
- **Rotary Positional Embeddings (RoPE)**: Positional information is encoded using rotary embeddings instead of absolute positional embeddings, enabling the model to better capture sequence-based patterns [42].

The LLaMA models are trained using the AdamW optimizer and tokenization part was handled by a Byte Pair Encoding (BPE) tokenizer, allowing the model to efficiently process complex and rare words, including multilingual text [42].

### 3.6 Mistral Models

The Mistral-7B-Instruct-v0.2 model, developed by Mistral, was the most effective model used in the fine-tuning experiments. It is a fine-tuned version of the Mistral-7B-v0.2 model, optimized through instruction tuning on publicly available datasets from Hugging Face [44]. Its performance has notably surpassed LLaMA-2 and LLaMA-3 models, as demonstrated in 1, especially in tasks requiring reasoning, code generation, and mathematics.

Mistral is based on the transformer architecture [45] but incorporates several improvements that make it more efficient, especially when working with long sequences. One of the key innovations is the Sliding Window Attention (SWA) mechanism, which limits each token's attention to a predefined window of previous tokens, reducing both the computation required and memory usage during inference [46]. This method allows the model to handle much longer sequences by attending only to a fixed number of tokens per layer, improving overall throughput. For example, at a window size of 4096 tokens, the model can theoretically attend to over 131,000 tokens across its layers which makes it highly effective for tasks involving long-context dependencies.

Another notable feature is the Rolling Buffer Cache, which helps manage memory more effectively during inference. Instead of storing all previous tokens, Mistral limits the cache size to match the window size, overwriting older tokens when necessary. This reduces the memory footprint for long sequences without impacting model accuracy, making it possible to efficiently process sequences that extend beyond typical cache limits [44]. The model also uses a pre-fill and chunking technique, which processes lengthy input prompts by breaking them into smaller chunks and pre-filling the cache for each chunk. This strategy allows Mistral to process long prompts in a memory-efficient manner, computing attention for both the cache and each chunk of input, thereby speeding up inference without compromising on the model's ability to capture context.

To further enhance efficiency, Mistral employs Grouped-Query Attention (GQA), which reduces the complexity of attention calculations [47]. By grouping queries, GQA enables faster inference, particularly for tasks requiring attention over longer sequences. This, combined with SWA, gives Mistral a considerable speed advantage over conventional transformer models [44].

Mistral uses a Byte-fallback BPE tokenizer, which divides text into subword units, improving its ability to handle both common and rare words across different languages [48]. This tokenizer enhances the model's understanding of linguistic diversity, making it

highly versatile for various NLP tasks [44].

The Mistral-7B model has been benchmarked against other models like LLaMA 2 13B, demonstrating superior performance across several categories, including code and reasoning benchmarks. Despite being a 7B parameter model, Mistral's performance is comparable to larger models, even outperforming some 13B models on certain tasks [44]. Additionally, the Mistral-7B-Instruct model, fine-tuned on instruction datasets, achieves high scores on benchmarks like MT-Bench.

## 3.7 Instruction-Tuning Large Language Models

Instruction tuning is a process designed to enhance the capabilities of pre-trained large language models (LLMs) by fine-tuning them on task-specific instructions which allows them to follow and respond to user prompts more effectively. This method capitalizes on the extensive pre-trained knowledge of LLMs while aligning them with specific tasks through instruction-based datasets. Rather than relying solely on supervised learning from labeled data, instruction tuning provides LLMs with explicit instructions on how to handle new unseen inputs [39].

### 3.7.1 LoRA

In this study, the fine-tuning framework deployed leverages the LoRA (Low-Rank Adaptation of Large Language Models) technique, as implemented through the Parameter-Efficient Fine-Tuning (PEFT) library from Hugging Face, alongside the SFFtrainer library designed for Supervised Fine-Tuning (SFT) [49] [50]. The dataset consists of natural language representations of customer profiles and their transaction histories, with the 10th merchant category purchase acting as the label for supervised learning.

LoRA is a parameter-efficient approach to model fine-tuning that preserves the pre-trained weights while focusing on the task-specific adaptation through low-rank parameterization [51]. Instead of directly updating the large pre-trained weights, LoRA introduces a pair of smaller trainable matrices that capture the task-specific knowledge. The methodology operates on the assumption that weight updates during fine-tuning have low intrinsic rank, and these updates can be effectively modeled as a low-rank decomposition

of the larger weight matrix.

The main update mechanism involves reparametrizing the weight matrix $W_0$ as follows:

$$W_0 = W + \Delta W,$$

where $W$ is the pre-trained weight matrix (kept frozen), and $\Delta W$ is the trainable update. However, $\Delta W$ is further decomposed into two smaller matrices:

$$\Delta W = AB^T,$$

where $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{d \times r}$ are the low-rank matrices, with $r \ll d$. This low-rank structure drastically reduces the number of parameters that need to be trained, where $r$ controls the rank and is typically much smaller than the original matrix dimensions. This decomposition allows LoRA to effectively model task-specific updates while using fewer parameters.

#### 3.7.1.1 Objective Function and Model Update Process

The learning objective, which optimizes the low-rank parameters, is represented by the following objective function:

$$\max_{\Theta} \sum_{(x,y) \in Z} \sum_{t=1}^{|y|} \log \left( P_{\phi + \Theta}(y_t \mid x, y_{<t}) \right).$$

Here, the model's parameters consist of the original, frozen pre-trained parameters $\phi$ and the low-rank updates $\Theta$. The goal is to maximize the log-likelihood of the target sequence $y = (y_1, y_2, ..., y_T)$ given the input sequence $x$ and the preceding tokens $y_{<t}$ across all input-output pairs in the dataset $Z$. Specifically, the log probability of each token $y_t$ is modeled as:

$$P(y_t \mid x, y_{<t}) = P_{\phi + \Theta}(y_t \mid x, y_{<t}),$$

where $P_{\phi + \Theta}$ represents the probability conditioned on both the frozen parameters $\phi$ and the low-rank updates $\Theta = AB^T$. During training, only the parameters in $A$ and $B$ are

updated, while $\phi$ remains unchanged, preserving the model's pre-trained knowledge [51].

#### 3.7.1.2 Low-Rank Decomposition in Transformers

In Transformer models, LoRA is primarily applied to the query and value projection matrices in the self-attention layers. These projection matrices, denoted as $W_Q$ and $W_V$, are typically large, and updating them directly would involve a significant number of parameters. However, by reparametrizing these matrices as $W_Q = W_Q + A_Q B_Q^T$ and $W_V = W_V + A_V B_V^T$, where $A_Q, B_Q, A_V, B_V$ are low-rank matrices, LoRA significantly reduces the number of parameters needed for fine-tuning.

The number of parameters to be trained is reduced from $d^2$ (for a full-rank matrix) to $2dr$, where $r$ is the rank of the low-rank matrices $A$ and $B$. This low-rank approximation maintains computational efficiency while ensuring that the model can still adapt to new tasks effectively [51].

A critical advantage of LoRA is that the original and low-rank matrices can be combined post-training into a single matrix during inference, ensuring that no additional latency is introduced. Specifically, during deployment, the reparametrized matrix $W_0 = W + AB^T$ is merged, allowing the model to behave as though it had been fully fine-tuned without the computational overhead of maintaining separate low-rank matrices.

This approach results in an efficient fine-tuning process that leverages the structure of the Transformer's attention mechanism, allowing it to adapt to new tasks using a minimal number of trainable parameters without sacrificing performance or introducing significant inference latency.

#### 3.7.1.3 Parameter Efficiency and Practical Impact

By keeping the majority of the pre-trained model's parameters frozen and updating only a small number of parameters through the low-rank matrices, LoRA achieves a highly efficient fine-tuning process. This process is particularly advantageous in scenarios where memory and computational resources are limited. Moreover, the method allows for task-specific knowledge to be incorporated into large pre-trained models without the risk of overfitting or losing the valuable general knowledge already captured by the original weights.

### 3.7.2 Parameter-Efficient Fine-Tuning Approaches

LoRA falls under a broader category of parameter-efficient fine-tuning (PEFT) techniques, which are designed to reduce the computational demands associated with adapting large models to new tasks. These techniques, including Adapters , Prefix Tuning, and Prompt Tuning, have become popular due to their ability to fine-tune models in a resource-efficient manner [52, 53]. Similar to LoRA, these methods typically freeze the majority of the model's parameters, updating only a small subset or introducing auxiliary parameters.

Adapters add small neural network modules between layers of the model that are trained during fine-tuning, while the original model weights remain unchanged. This method allows for task-specific fine-tuning with minimal additional computation [52] that are used in techniques such as Prefix Tuning and Prompt Tuning [53].

Each of these techniques, including LoRA, aims to mitigate the cost of fine-tuning large models while retaining high performance, making them ideal for scenarios where computational resources are limited, or the model needs to be rapidly adapted to new domains.

## 3.8 Dataset

A major financial institution in an OECD country donated two deidentified samplings of their data (called as Bank A throughout this study) that were collected over the period between July 2014 and July 2015 [54]. The dataset presented comprises a range of attributes that offer a multifaceted view of customer transactions. Each entry includes a masked customer identifier to maintain privacy while allowing for the tracking of individual customer activities.

| musteri_id_mask | islem_tarihi | islem_tutari | cinsiyeti | medeni_drm_ack | egitim_drm_ack | is_turu_ack | gelir | yas | category_name |
|---|---|---|---|---|---|---|---|---|---|
| 3867546 | 2014-08-05 | 460 | E | EVLI | LISE | SERBEST MESLEK | 12000 | 54 | Erkek ve Kadın Giyim Mağazaları |
| 12912748 | 2015-02-11 | 288 | K | BEKAR | ÜNIVERSITE | ÜCRETLI (ÖZEL) | 5000 | 29 | Bakkallar ve Süpermarketler |
| 25514011 | 2015-02-04 | 830 | K | EVLI | ÜNIVERSITE | ÜCRETLI (ÖZEL) | 2000 | 27 | Erkek ve Kadın Giyim Mağazaları |
| 8402718 | 2014-10-12 | 479 | K | BOSANMIS | DOKTORA | ÜCRETLI (ÖZEL) | 40000 | 59 | Servis İstasyonları |
| 4785643 | 2014-10-12 | 100 | K | BEKAR | DOKTORA | ÜCRETLI (ÖZEL) | 40000 | 59 | Bakkallar ve Süpermarketler |

Table 3.1 A sample transaction raw data from Bank A, showing how raw data looks like before preparing it for fine-tuning

The table 3.1 shows a sample of customer transaction data from Bank A, where each row corresponds to an individual customer's specific transaction. The features capture various characteristics of both the transaction and the customer. The musteri_id_mask feature

serves as a masked customer identifier, ensuring data privacy while allowing tracking of transactions. The islem_tarihi feature records the date on which the transaction occurred, while the islem_tutari feature represents the monetary amount involved in the transaction.

The cinsiyeti feature provides the gender of the customer (denoted by "E" for male and "K" for female). Medeni_drm_ack, referring to marital status, specifies whether the customer is married ("EVLİ"), single ("BEKAR"), or divorced ("BOSANMIS"). The egitim_drm_ack feature reflects the education level of the customer, categorized as "Lise" (high school), "Üniversite" (university), or "Doktora" (Ph.D.), providing insight into the demographic profile.

The is_turu_ack feature, which refers to the type of employment, differentiates customers by their work status, such as self-employed ("Serbest Meslek") or salaried in private sectors ("Ücretli Özel"). The gelir feature represents the customer's income, offering a key financial indicator, while yas denotes the customer's age. Lastly, the category_name feature classifies the transaction by the type of establishment, such as supermarkets, clothing stores, or service stations.

Bank A dataset consists of millions of transactions conducted by 10,000 distinct customers, involving 482 different merchants over a time span from July 2014 to June 2015. The majority of these transactions took place in Istanbul, one of Turkey's major metropolitan areas. Each merchant is assigned a specific category, reflecting the type of business or service they provide. These merchant categories include: Service Stations, Houseware Shops, Grocery Stores and Supermarkets, Direct Marketing and Insurance Services, Man and Woman Clothing Shops, Electronics Shops, Other Commercial Activities, Public Services, Telecommunication Services, Sports Shops, Insurance Sale, Car Parks and Garages, Food Places and Restaurants, Airlines, Travel Agents and Tour Operators, Hospitals, Cosmetics Shops, and Bus Routes.

Another dataset from a different financial institution referred to as Bank B in this study, was utilized exclusively for testing purposes. While the Bank A dataset was employed in both the training and testing phases, the Bank B dataset was introduced to evaluate the performance of the models trained on Bank A data in a completely different context. The Bank B dataset consists of tens of thousands of individual customer accounts, which represent approximately 10% of the total customer accounts in the data warehouse of a major financial institution in an OECD country [55].

Each customer account in the Bank B dataset contains detailed data on all credit card transactions made for purchases over a three-month period in 2013, specifically from April to June. The dataset includes over 10 million transactions during this time frame and the

features present in the Bank B dataset are identical to those in the Bank A dataset. This parallel structure between the two datasets allows for a seamless comparison of model performance across different banks and timeframes.

## 3.9 Data Preprocessing

### Merchant Category Filtering

The first step in data cleansing involved filtering transactions to include only those from the three most frequently purchased merchant categories: Grocery Stores, Clothing Stores, and Gas Stations. This decision was motivated by the need to address class imbalance, as underrepresented categories such as insurance could introduce bias during model training. The three categories selected—Grocery Stores, Clothing Stores, and Gas Stations—represented the most transactionally active sectors within the dataset. To preserve the natural transactional sequence and behaviors, merchant categories outside of these top three were labeled as "Other" and included in both the training and testing phases as a distinct category. This approach ensured that the full range of customer behaviors was captured without overrepresenting any single minor category.

### Handling Missing and Incomplete Data

To ensure a high-quality dataset, users with incomplete demographic information, including gender, marital status, education level, employment type, income, and age, were excluded. Specifically, the rows with null values in features of gender, marital status, education level, employment type, income, age, merchant category, transaction date, and transaction amount were dropped to maintain the comprehensiveness and reliability of the data. This ensured that the dataset used for model training was not only representative of real-world behaviors but also complete and free from missing values that could potentially lead to skewed outcomes.

### Transaction Threshold and Category Diversity

Additionally, users with fewer than ten transactions across at least two distinct categories were excluded. This filtering was applied to eliminate users with insufficient transactional histories, such as truck drivers frequently purchasing from gas stations, which would introduce noise into the model and bias predictions toward a single merchant category. By enforcing a minimum threshold for both transaction count and category diversity, the dataset was refined to focus on users with more robust and varied transactional behaviors.

### Standardization of Merchant Category Names

In further preprocessing, merchant category names were standardized to reduce the number of unique values. For instance, the dataset originally contained many variants of merchant categories, such as "Erkek ve Kadın Giyim Mağazaları" (Men's and Women's Clothing Stores) and "Bakkallar ve Süpermarketler" (Grocery Stores), which were mapped to a common set of three values. Any category not belonging to the main three categories—Grocery Stores, Clothing Stores, and Gas Stations—was consolidated under the "Other" category to simplify classification while preserving relevant transaction data for model fine-tuning.

### Demographic Data Processing

To provide additional demographic insight, an *income group* column was created by dividing the income variable into three quantiles, representing low, middle, and high-income groups. This allowed for further analysis of purchasing behaviors across different socioeconomic segments.

The demographic variables were further processed by standardizing categorical values. Gender was normalized to "male" and "female," while marital status was mapped to values such as "married," "single," "divorced," "widowed," and "unknown." Similarly, education level feature was standardized to categories like "high school," "university," and "graduate," while employment type was harmonized across different entries, mapping to values such as "private employee," "self-employed," and "retired."

### Dataset Size Reduction and Refinement

Through these preprocessing steps, the dataset was reduced from an initial pool of 10,000 users and 298 categories, encompassing over one million transactions, to a refined cohort of 8,154 users and four consolidated categories—Grocery, Clothing, Gas Stations, and Other—comprising 1,123,445 transactions. These measures ensured the data was well-structured, representative, and ready for the fine-tuning of the language model. This rigorous preprocessing was critical in ensuring that the model training process would yield accurate and generalizable predictions based on high-quality, well-defined transactional data.

### Dataset Partitioning for Model Training

To evaluate the performance of the model during training and ensure its generalization to unseen data, the dataset was partitioned into training, validation, and test sets using an 80-10-10 split. Initially, 80% of the data was designated for training, while the remaining 20% was split equally between validation and test sets. Specifically, the *train_test_split* function from the scikit-learn library was employed to divide the data into a training set

and a temporary set, comprising 20% of the total data. This temporary set was then further split into validation and test sets, each accounting for 10% of the total dataset. A random seed of 42 was used to ensure reproducibility. The resulting subsets were converted into the Hugging Face *Dataset* format to enable their use in model fine-tuning. This partitioning strategy ensured that the validation set was used for model tuning, while the test set remained independent for final performance evaluation.

## 3.10 Finetune Dataset Preparation

In transforming the customer data from its original tabular format, as seen in Table 3.1 above, into a dataset suitable for instruction-tuning a language model, a systematic and structured approach was adopted. The goal was to articulate each customer's data in a natural language style to facilitate instruction tuning. This involved a four-step process to prepare the data for model fine-tuning, which aimed to predict a customer's future purchase category based on historical transaction data and demographic details [39, 5].

**Raw Data Transformation to Natural Language Format and Prompting**

The first step in the transformation process involved converting the raw transaction data into a natural language format, which served as the foundation for creating the fine-tuning dataset. This transformation was based on an instruction-tuning strategy that emphasized making the data more relatable by using a first-person narrative style. The goal was to simulate a scenario where the user was describing their own transaction history and demographic details, thereby making the input to the language model more conversational and contextually grounded.

The next step in the data preparation process involved translating demographic attributes from Turkish to their English equivalents and standardizing the terminology. This translation was accomplished using a series of custom basic mapping dictionaries that converted Turkish terms for gender, marital status, education level, and employment type into corresponding English labels. Specifically, the code utilized predefined mappings to consistently replace Turkish terms with their English counterparts, while accounting for variations in spelling, case, and formatting. For instance, gender values like "E" and "K" were mapped to "male" and "female," respectively, and marital status terms such as "evli" (married) and "bekar" (single) were translated accordingly. Similarly, education levels like "LİSE" or "ÜNİVERSİTE" were standardized to "high school" and "university," while various terms for employment types (e.g., "serbest meslek" as "self employed"

or "ücretli (özel)" as "private employee") were translated and unified. Additionally, the mappings accounted for inconsistent cases (uppercase, lowercase) and extra spaces to ensure uniformity across the dataset. The standardized values were then applied to the data, replacing the original Turkish entries with their corresponding English translations to make the dataset more suitable for LLMs that were originally trained on large english corpus.

In the data preparation process, the mapping of merchant category names aimed to reduce the number of unique values by consolidating similar categories into three primary groups: "Grocery," "Clothing," and "Gas Stations." To achieve this, a custom mapping function was implemented to standardize the original Turkish merchant category names by matching them against a predefined dictionary. This dictionary mapped various related categories, such as "Bayan Hazır Giyim Dükkanları" (Women's Clothing Stores) and "Aile Giyim Mağazaları" (Family Clothing Stores), to a unified category, "Erkek ve Kadın Giyim Mağazaları" (Men's and Women's Clothing Stores). The function also accounted for inconsistencies like extra spaces, spelling variations, and different cases by normalizing the category names to lowercase before mapping them. If a category was not found in the mapping dictionary, it was added to a list of unmapped categories.

Following the mapping, the data was further refined by translating the consolidated categories into English equivalents for consistency. Categories mapped to "Bakkallar ve Süpermarketler" (Grocery Stores), "Erkek ve Kadın Giyim Mağazaları" (Clothing Stores), and "Servis İstasyonları (Asistans-Yardım Servisi Olan veya Olmayan)" (Service Stations) were translated to "Grocery," "Clothing," and "Gas Stations," respectively. All other categories were grouped under "Other" to simplify the dataset while preserving the transactional sequence of customers.

In the construction of the prompt, the narrative style was structured as if the user was explaining their own profile and purchase history. This approach involved describing the demographic and historical transaction information using natural language expressions such as "I am," "I bought," and "I spent." For instance, the prompt included details like the user's age, gender, marital status, education level, employment status, income group, and a summary of recent transactions. The prompting design aimed to provide a comprehensive contextual representation while maintaining simplicity and relevance.

The following pseudocode illustrates the general process of preparing the input data, where data points are selected based on certain criteria (e.g., recent transactions, variety in purchase categories). The data preparation involved calculating aggregates like total spending and checking for sufficient diversity in purchase categories to ensure informative instruction tuning examples.

```
# Pseudo-code for preparing training data for instruction tuning
For each customer:
    Sort transactions by date and select the last "n" transactions.
    If less than "n" transactions, skip to the next customer.
    Extract customer details for demographic information:
        - Age, gender, marital status, education level, employment status
        - Income group and other relevant features
    Create strings for purchase categories, dates, and amounts:
        - List of categories for the "n-1" transactions
        - Corresponding dates and amounts
    Compose a natural language input string:
        - "I am <customer_id>. I am <age> years old, <marital_status> <gender>,
          <education_status> graduate, and <employment_status>. In terms of my income state,
          I belong to the <income_group> income group. Recently, I made <n-1> transactions.
          I bought items from the following categories, chronologically: <list_of_categories>.
          I bought from these categories on the following dates, chronologically: <list_of_dates>.
          I spent the following amounts, chronologically: <list_of_amounts>. Total spent: $<total_amount>."
    Assign the target output as the category for the "n"-th transaction.
    Store the "Instruction Input" and "Instruction Output" pairs in json.
```

Listing 3.1 Pseudo-code for preparing data in prompt style for instruction tuning for each customer

This pseudocode in Listing 3.1 ensures that the prompt format captures essential information in a natural language style which simulates a personalized description by the user.

### Instruction and Response Formatting

The natural language description is structured into an "Instruction Input" and "Instruction Output" pair. The process involves four key components:

1.1 **Task Instruction:** The task is defined through a natural language instruction in prompting style, such as: "Based on my demographic details and historical transaction data provided below, predict my next purchase category." This concise instruction encapsulates the nature of the prediction task that this study attempts

solving.

1.2 **Task Input:** The input combines the customer's demographic information and the details of their last "n-1" transactions, formatted as described in the previous subsection.

1.3 **Task Output:** The output represents the expected category of the "n"-th purchase, providing the target for the model's prediction.

1.4 **Instruction Pair Construction:** The "Task Instruction" and "Task Input" are synthesized to form the "Instruction Input," while the "Task Output" is set as the "Instruction Output." Together, these form the dataset sample used for tuning.

**Data Preparation for Model Training in Mistral Style**

After creating the "Instruction Input" and "Instruction Output" personalized pairs for each customer, further formatting is required to prepare the data for model training. This step includes adding special tokens and organizing the data in a structure suitable for the training framework. Below is a pseudocode example illustrating this process:

```
# Pseudo-code for formatting data before model training in Mistral Style

For each "Instruction Input" and "Instruction Output" pair:
    Add special tokens:
        - "<s>[INST]" at the start of the instruction
        - "[/INST]" at the end of the instruction
        - "</s>" at the end of the response

    Combine the instruction and response into a single formatted string:
        - "<s>[INST] <instruction> <task_input> [/INST] <task_output></s>"

Store the formatted strings in a list for further processing

Write the formatted data to a text file, separating each entry with a
    newline
```

Listing 3.2 Pseudo-code for formatting data before Mistral model training

This formatting step ensures that the data is in a structure that allows for effective fine-tuning. This type of formatting is highly linked to the way the pretrained instruction model was trained with a specific prompting for creating chat template. Chat templates like 3.2 are not compulsory but yet it is critical during training because it provides a good way to ensure that the chat template matches the tokens the model sees during training. Hence, these tokens were the ones that were used while scientists behind Mistral model

was pretraining the model for instruction tuning [56]. They help the model recognize the boundaries of the instruction and response, which is critical for distinguishing between the input prompt and the expected output during training.

### Data Preparation for Model Training in Alpaca Style for LLaMa Models

The Alpaca-style data preparation follows a distinct approach from the Mistral-style formatting and it is tailored specifically for fine-tuning LLaMa models. The preparation steps ensure that the model can effectively interpret the relationship between tasks, inputs, and expected outputs. The pseudocode below demonstrates the process of transforming already processed data in Listing 3.1 into a format suitable for model training of LLaMa models using Alpaca style instruction:

```
Function format_data(input_file, output_file):
    Load data from the input JSON file
    Create an empty list for formatted data
    For each item in the data:
        instruction = item['instruction']
        input_text = item['input']
        output_text = item['output']
       # Format using the Alpaca-style template
        formatted_text = (
            "Below is an instruction that describes a task, paired with an input that provides further context. "
            "Write a response that appropriately completes the request.\n"
            "### Instruction:\n" + instruction + "\n"
            "### Input:\n" + input_text + "\n"
            "### Response:\n" + output_text
        )
        Add formatted_text to the formatted data list
    Write each entry in the formatted data list to the output text file, one per line

Function load_and_process_data(file_path):
    Read all lines from the specified text file
    Create an empty list for processed data
    For each line:
        If the line is not empty:
            Clean the line
            Add {'text': cleaned_line} to the processed data list
    Convert the processed data list to a DataFrame
    Transform the DataFrame into a dataset compatible with the training
```

```
    framework
28  Return the prepared dataset
```

Listing 3.3 Pseudo-code for formatting Alpaca-style data for LLaMa models

The data preparation process involves several key steps. It begins with loading raw data from a JSON file coming from processing step as shown in 3.1, where each text entry consists of fields such as instruction, input, and output. The code then iterates over each item in the dataset, extracting these fields and formatting them into a specific prompt structure. This structure is designed to simulate real-world instruction-following scenarios, where an instruction describes the task, an input provides additional context, and a response completes the task. The formatted text includes special tokens such as `### Instruction` `### Input` and `### Response` to clearly distinguish these parts within the training samples. The prompt `"Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request"` is used at the beginning of each instruction-input-output pairs. Each formatted entry is stored in a list, which is later written to an output text file, ensuring that each line of the file represents an individual training instance (which are personalized prompts for each customer) in the Alpaca format.

The dataset is then split into training, validation, and test sets to enable proper evaluation of the model's performance. A portion of the data is used for validation to tune hyperparameters, while the remainder is reserved for final testing.

The format used for instructions in Alpaca format follows a specific prompt structure:

```
1 Below is an instruction that describes a task, paired with an input that
    provides further context. Write a response that appropriately completes
    the request.
2 ### Instruction:
3 {instruction}
4 ### Input:
5 {input}
6 ### Response:
```

This structure highlights the separation between the instruction, input, and response, making it easier for the model to understand the context of each task. Unlike the Mistral-style formatting, which relies on special tokens to delineate components, the Alpaca-style format uses explicit headers for each data sample sections of instruction, input, and output.

### Dataset Preparation for Training Framework

The prepared text data (whether it is Alpaca or Mistral format) is then loaded into a suitable framework for fine-tuning. In this case, the dataset is converted into a format compatible with the Hugging Face `datasets` library, which supports various NLP tasks. The dataset is split into training, validation, and test sets for model evaluation:

```
# Pseudo-code for dataset preparation
Load the formatted data from the text file
Split the data into train and test sets (e.g., 80% training, 20% testing)
Further split the test set into validation and test sets (50% each)

Store the dataset splits in a dictionary:
    - "train": training set
    - "validation": validation set
    - "test": test set
```

Listing 3.4 Pseudo-code for dataset preparation

This procedure enables efficient organization of the dataset for model fine-tuning while ensuring that the model's performance can be assessed on separate validation and test sets.

Table 3.2 A sample format created from the raw transaction dataset for fine-tuning.

| **Instruction Input** |
|---|
| **Task Instruction:** Based on my demographic details and historical transaction data provided below, predict my next purchase category.<br>**Task Input:** I am 1695432. I am 48 years old, married male, secondary school graduate, and I am working as a private employee. In terms of my income state, I belong to the high-income group. Recently, I made 9 transactions. In these transactions, I have spent a total of $560.61 dollars. I bought items from the following categories, chronologically: Grocery, Grocery, Grocery, Other, Clothing, Other, Clothing, Clothing, Clothing. I bought from these categories on the following dates, chronologically: 2015-03-28, 2015-04-01, 2015-04-15, 2015-05-01, 2015-05-27, 2015-06-04, 2015-06-04, 2015-06-04, 2015-06-08. I spent the following money for these items, chronologically: $39.82, $47.25, $27.81, $124.97, $105.97, $24.95, $49.99, $99.95, $39.90. |
| **Instruction Output** |
| **Task Output:** Gas stations. |

The following Table 3.2 illustrates a single formatted instruction tuning data sample that

is feed into model during training. This example demonstrates how the raw data in Table 3.1 is transformed into a narrative format, incorporating structured input and response pairs to facilitate effective fine-tuning of the language model.

## 3.11 Training Steps

The fine-tuning was carried out through a systematic procedure that adapted the model's weights to predict customer purchase categories based on historical transaction data and demographic information. This section details the training process, including data loading, model configuration, training setup, and monitoring. The steps described here were implemented for the purpose of this study to fine-tune a pre-trained language model using the instruction-based input-output pairs prepared in the dataset preparation phase above.

- **Model and Tokenizer Setup**
  - The model and tokenizer (either LLaMa instruct or Mistral instruct models) were initialized using pre-trained checkpoints downloaded from huggingface model hubs. The models used were causal language models, loaded with `AutoModelForCausalLM.from_pretrained` from the `transformers` library.
  - To optimize memory efficiency, a quantization configuration was applied using `BitsAndBytesConfig`, which loaded the model in 4-bit precision with NF4 quantization. This setup reduced memory consumption.
  - The tokenizer was configured using `AutoTokenizer.from_pretrained`, with the padding token set to match the end-of-sequence token to ensure consistent input handling.
- **Configuration of PEFT (Parameter-Efficient Fine-Tuning)**
  - The fine-tuning process employed Parameter-Efficient Fine-Tuning (PEFT) techniques, specifically Low-Rank Adaptation (LoRA), to adjust only a subset of the model's parameters.
  - The LoRA configuration, defined via `LoraConfig`, included parameters such as `r`, `lora_alpha`, and `lora_dropout`, which controlled the rank, scaling factor, and dropout rate for the adapted layers.

  - Target modules for adaptation were specified, focusing on components such as `q_proj`, `k_proj`, `v_proj`, and `o_proj` within the model's attention mechanism to allow effective fine-tuning with minimal computational cost.

- **Training Arguments and Setup**
  - The training setup was defined using `TrainingArguments` from the `transformers` library, which specified parameters controlling the training process.
  - Mixed precision training was enabled with the `bf16` flag, using bfloat16 precision to accelerate computations without compromising accuracy.
  - The learning rate was managed with a cosine schedule, and a warmup ratio was applied to increase the learning rate gradually during the initial phase.
  - Logging was performed at intervals specified by the `logging_steps` parameter, and evaluation was carried out periodically based on the `evaluation_strategy` setting. TensorBoard was used for real-time monitoring of the training progress.

- **Data Loading**
  - The dataset, prepared as described in the earlier sections, was split into training, validation, and test sets to allow for model evaluation during and after training.
  - The `datasets` library facilitated efficient handling of NLP datasets, and a `DataLoader` from PyTorch was used to create data iterators for training.
  - The `max_seq_length` parameter controlled the maximum sequence length for input data, ensuring that input sequences were appropriately truncated or padded.

- **Training Procedure Using the SFTTrainer** The training was executed using the `SFTTrainer`, a custom trainer class designed for fine-tuning language models with instruction-based tasks. The following steps were performed during training.
  - *Training:* The `train()` method from PyTorch was used, which set the model to training mode. It enables activating dropout and using batch statistics in batch normalization layers to prevent overfitting and facilitate learning.
  - *Batch Processing:* Batches of tokenized data were fed into the model, consisting of instruction-input pairs formatted as previously described.

- *Loss Computation:* The model computed the loss between the predicted output tokens and the target tokens, using cross-entropy loss as the criterion.
- *Gradient Accumulation and Backpropagation:* Gradients were accumulated over multiple steps (as defined by `gradient_accumulation_steps`) before updating the model parameters.
- *Parameter Updates:* The optimizer used was Paged AdamW, and parameters were updated according to the learning rate schedule.
- *Checkpointing:* The model was saved at specified intervals (e.g., at the end of each epoch) to allow for resuming training or performing inference later.
- *Monitoring:* Training progress was monitored using TensorBoard, providing insights into metrics such as loss and learning rate.

- **Post-Training and Model Saving**
  - After training, the fine-tuned model was saved using `trainer.model.save_pretrained()`, storing the fine-tuned parameters and checkpoints for future use.
  - The saved model incorporated the knowledge acquired during fine-tuning, making it suitable for inference tasks as described in the subsequent sections.

The training process described above leveraged efficient techniques such as PEFT and mixed precision training, with monitoring enabled through TensorBoard to ensure a scalable fine-tuning workflow.

## 3.12 Loading the Fine-Tuned Model for Inference

Once the fine-tuning process is complete, the next step is to load the fine-tuned model on top of a pre-trained base model for inference. This approach allows leveraging the adaptations learned during fine-tuning while utilizing the foundational knowledge of the pre-trained model. The Parameter-Efficient Fine-Tuning (PEFT) framework is employed to achieve this and to enable the loading of the fine-tuned model along with the original base model [49].

The general procedure for loading the model includes the following steps that this study implemented:

2.1 **Initializing the Base Model:** The base language model in its pre-trained form is loaded. This model serves as the foundation upon which the fine-tuned parameters are applied. The loading process may involve specifying memory management configurations, such as the data type for weights and device allocation.

2.2 **Loading the Fine-Tuned Adapters:** Using PEFT's `from_pretrained` method which adds a small number of trained parameters (the adapters) on top of the pre-trained model [57], loading the fine-tuned model parameters, referred to as adapters, on top of the pre-trained model is a necessary step. These adapters contain the additional weights or modified configurations learned during fine-tuning. After loading, the adapters can be merged into the base model.

2.3 **Tokenizer Configuration:** The tokenizer corresponding to the base model is loaded, and any necessary modifications are made (e.g., setting special token identifiers). This step ensures consistency between the model and the tokenization process used for input preparation.

2.4 **Special Tokens Setup:** Define the stopping criteria or special tokens that indicate the end of the generated output. This configuration helps control the generation process during inference.

This process ensures that the fine-tuned model is loaded effectively, combining the knowledge of the pre-trained base model with the specific tuning performed during fine-tuning. It allows the model to perform the desired inference tasks based on the learned parameters.

## 3.13 Inference Using the Fine-Tuned Model

Inference involves generating predictions from the fine-tuned model using natural language input prompts structured according to the fine-tuning format. During inference, the model is provided with instruction-based input, and it generates an output that aligns with the expected response format. The inference process can be broken down into several steps:

3.1 **Input Preparation:**

- The input data consists of an instruction followed by additional contextual information relevant to the task. The output is not given because it is the

ground truth, or the expected response that is expected from our trained model to predict.

- The text input is formatted to match the prompt style used during fine-tuning, just like in the Table 3.2. In cases where specific markers or delimiters are used (e.g., special tokens like [INST]), they are appended to the input to maintain consistency with the training setup.

3.2 **Tokenization:**

- The formatted input text is tokenized using the tokenizer associated with the fine-tuned model. This process converts the input text into a sequence of token identifiers that can be processed by the model.
- The length of the input sequence is recorded to differentiate between the prompt and the generated output during the decoding phase.

3.3 **Model Inference:**

- The tokenized input is fed into the model, where it undergoes forward processing to generate a sequence of token predictions via using `torch.inference-mode()` method from PyTorch that acts as a context manager [58]. The logic behind this manager is to disable several functionalities not being necessary for generation like gradient tracking to make forward-passes (data going through the forward() method) faster. The generation process may involve controlling factors such as the maximum length of the output sequence and the temperature parameter to influence the randomness of the output.
- The generated tokens beyond the input length are considered as the output response, representing the model's prediction for the given task.

3.4 **Post-Processing:**

- The generated token sequence is decoded back into natural language text by skipping any special tokens used during processing.
- The output text is encoded using basic LabelEncoder from scikit learn library and compared against the expected response (which is encoded similary to actual response) to evaluate the model's accuracy and performance.

The inference procedure for models fine-tuned with different styles (e.g., Mistral or Alpaca) follows the same general framework but involve variations in input formatting steps as explained in Mistral and Alpaca style data formatting in above pseudo-code. In both

cases, the fine-tuned model uses the provided instruction and contextual input to generate the desired response. Later, the generated responses are compared with the ground truths to get the final performance results.

# 4. Evaluation

The task at hand is to predict user preference among four categories (in which one of them was labeled as 'Other'), which constitutes a multiclass classification problem. To this end, a suite of standard evaluation metrics is utilized by following the current literature, comprising accuracy, precision, recall, and the weighted F1 score. These metrics provide a holistic view of the model's performance by not only considering the proportion of correct predictions (accuracy) but also the models' ability to correctly identify positive cases (precision and recall) and a weighted measure of precision and recall (weighted F1 score).

## 4.1 Performance Metrics

In this study, the selection of evaluation metrics is centered around the need to provide a balanced and nuanced assessment of model performance, particularly in the context of imbalanced datasets. While several metrics are employed, the **Weighted F1 score** emerges as the most critical metric due to its ability to account for both precision and recall while adjusting for class imbalances, making it the primary evaluator in determining model effectiveness.

- **Overall predictive success**, measured by **Accuracy**, offers a high-level indication of the model's performance across all classes. However, in datasets with class imbalances, accuracy alone can be misleading, as it may disproportionately reflect the performance of majority classes.
- **Precision** and **Recall** provide a more granular view of the model's ability to correctly identify instances for each class, especially important in imbalanced datasets where false positives or false negatives can distort the overall picture. Precision measures the model's accuracy in predicting a particular class, while recall mea-

sures the model's ability to capture all relevant instances of that class.

- The **Weighted F1 score**, however, serves as the most decisive metric in this study. By harmonizing precision and recall into a single value and weighting it according to the distribution of classes, the weighted F1 score ensures a more comprehensive evaluation. It accounts for the imbalanced representation of classes, ensuring that the performance in underrepresented categories like *Clothing* and *Gas Stations* is not overshadowed by majority classes such as *Grocery*. This makes it the most robust measure for evaluating model effectiveness across all categories, especially when assessing generalization to unseen data.

The emphasis on the weighted F1 score is critical in the context of this research, where accurately predicting user preferences in all merchant categories—both majority and minority—is essential. While accuracy provides a broad sense of success, it is the weighted F1 score that offers the most reliable insight into a model's balanced performance across the entire dataset. This metric was ultimately the primary decision-maker in evaluating the success of models experimented in this study. The subsections below provide detailed definitions and mathematical formulations of these metrics.

### 4.1.1 Accuracy

Accuracy is one of the simplest and most intuitive performance metrics for classification problems. It is defined as the ratio of the number of correct predictions to the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Where:

- $TP$ (True Positives): Correctly predicted positive cases.
- $TN$ (True Negatives): Correctly predicted negative cases.
- $FP$ (False Positives): Incorrectly predicted as positive.
- $FN$ (False Negatives): Incorrectly predicted as negative.

Accuracy works well when the dataset is balanced, but in cases of class imbalance like

the dataset that this study deals with, it may give a misleading sense of performance.

### 4.1.2 Precision

Precision focuses on the proportion of true positive predictions relative to the total number of positive predictions. It tells how many of the predicted positive cases are actually correct.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

High precision indicates that the model returns more relevant results (i.e., fewer false positives), making it particularly important when the cost of false positives is high.

### 4.1.3 Recall (Sensitivity)

Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify all relevant instances in the dataset. It is the ratio of true positives to the actual total number of positive cases.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

High recall is essential in cases where missing a positive case (false negative) is more costly or harmful, as it captures the model's ability to detect all true positives.

### 4.1.4 F1 Score (Weighted)

The F1 score is the harmonic mean of precision and recall. It provides a single measure that balances the trade-off between precision and recall, especially in cases where the

class distribution is uneven.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

For multiclass classification, the weighted F1 score is often used to take the class imbalance into account. The weighted F1 score averages the F1 scores for each class, weighted by the number of true instances in each class:

$$\text{Weighted F1} = \frac{\sum_c w_c \cdot F1_c}{\sum w_c} \tag{4.5}$$

Where $w_c$ is the number of true instances in class $c$.

The weighted F1 score provides a more comprehensive assessment when class imbalance exists, as it reflects how well the model performs across all classes.

## 4.2 Experiments

The chosen LLMs are distinguished by their pre-existing instruction tuning on task-oriented activities, their accessibility as open-source weights, their documentation quality, their commendable performance in benchmarks pertinent to natural language processing tasks, and their recognition within the open-source LLM community. Each model was fine-tuned using a dataset comprising 8154 unique customers, representing 90% of the total dataset. For this training, a single Nvidia RTX 3090 with 24 GB RAM was used.

The first two models subjected to fine-tuning via the Alpaca method (which is a fine-tuning dataset formatting method that is a little different than the main fine-tune dataset formatting explained above) were the LLaMA-2-Chat 7B and LLaMA-3-8B instruct, auto-regressive language models utilizing an optimized transformer architecture [59, 60, 43]. This model has been further refined by Meta through both supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF), with a focus on alignment with human preferences for helpfulness and safety. The architecture enhancements enable the model to engage in dialogue with improved responsiveness and ethical awareness.

The third and the most efficacious model in the fine-tuning experiments was the Mistral-7B-Instruct-v0.2 created by Mistral. This model is a fine-tuned iteration of the Mistral-7B-v0.2 and has been optimized using instruction tuning on publicly available datasets hosted on the Hugging Face platform [44]. Noteworthy is its superior performance over the LLaMA 2 and LLaMA 3 models as demonstrated in Table 1 and Table 4.4. The Mistral-7B-Instruct-v0.2 employs Grouped-query attention (GQA) for accelerated inference and Sliding Window Attention (SWA) for efficiently managing longer sequences. Moreover, it utilizes a Byte-fallback BPE tokenizer to enhance its capability to process and understand a diverse range of linguistic inputs.

Each model's architecture and training history contribute to its unique strengths in processing and predicting natural language patterns. The fine-tuning exercises conducted as part of this research were aimed at leveraging these strengths to predict the transactional behavior of customers. Naturally, this is a task that necessitates a deep understanding of both language and human purchase behavior. The results from these experiments were expected to provide insights into the models' adaptability and performance in the specific context of merchant category prediction.

In the testing inference phase, various input sequence lengths were employed to rigorously assess the models' generalizability and to provide a comprehensive range of performance scores. Despite being trained on the last nine transactions, the models were tested with sequences of four, seven, and fourteen transactions.

### 4.2.1 LLM Training Performance and Hyperparameters

The three figures presented offer valuable insights into the training and evaluation loss for the Llama2, Llama3, and Mistral models over a series of training steps which highlights the convergence behavior and the stability of each model during fine-tuning.

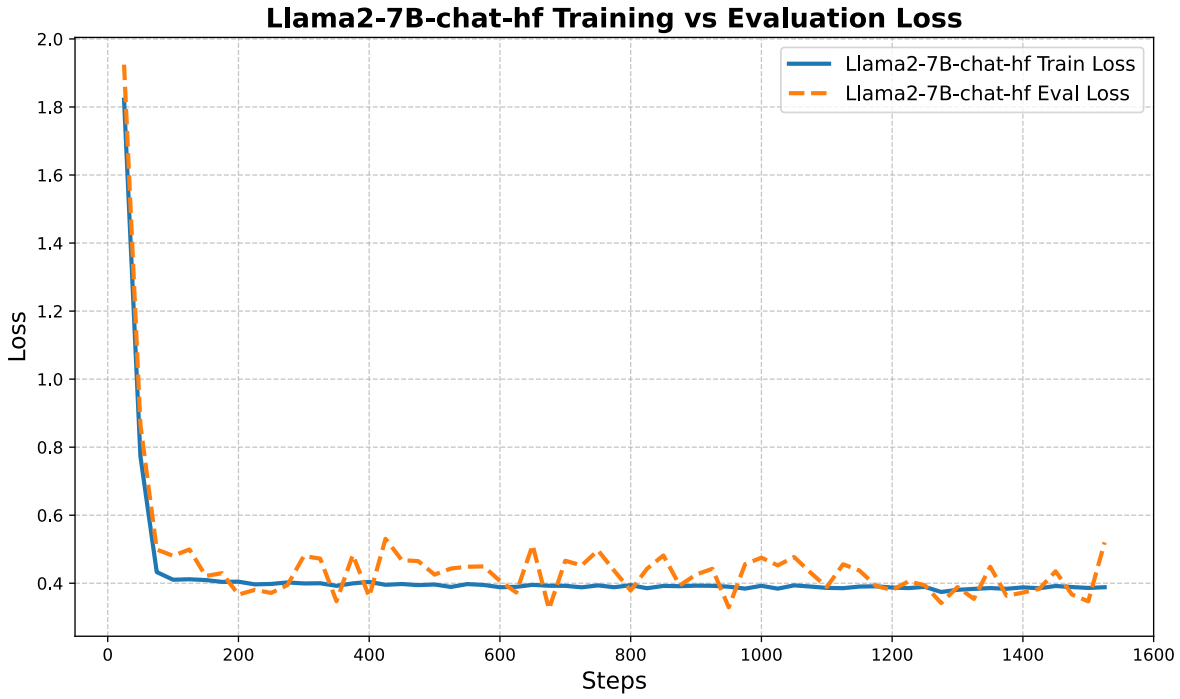


Figure 4.1 Training vs evaluation loss for the Llama2 model.

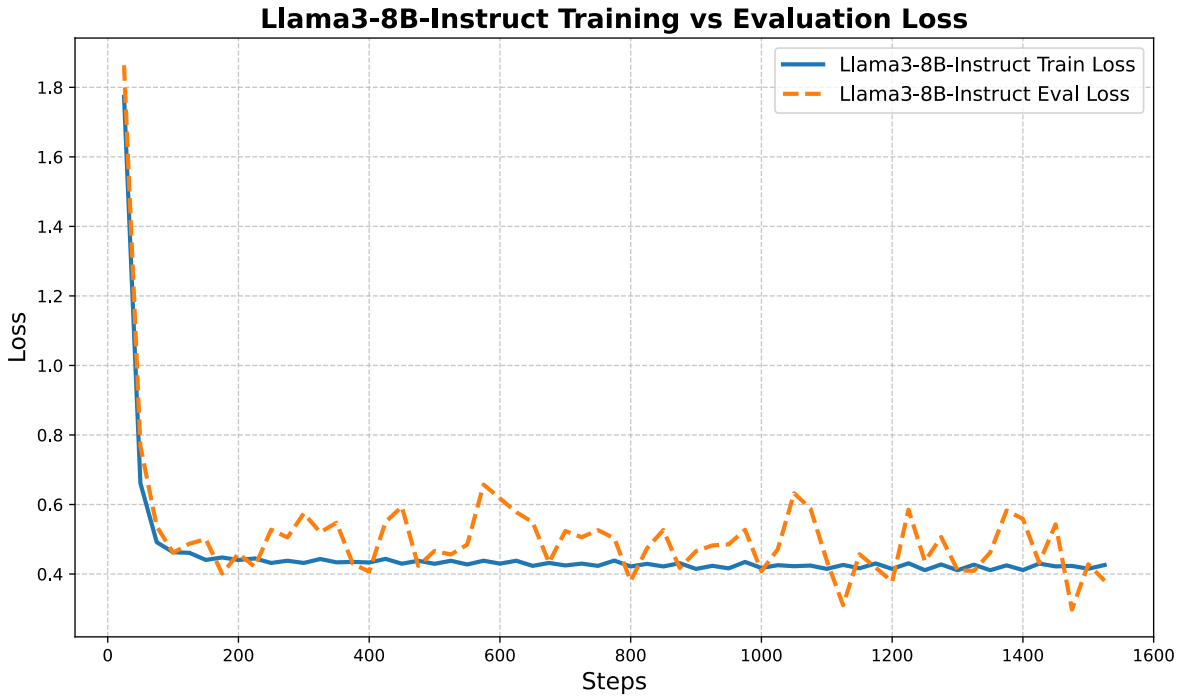


Figure 4.2 Training vs evaluation loss for the Llama3 model.

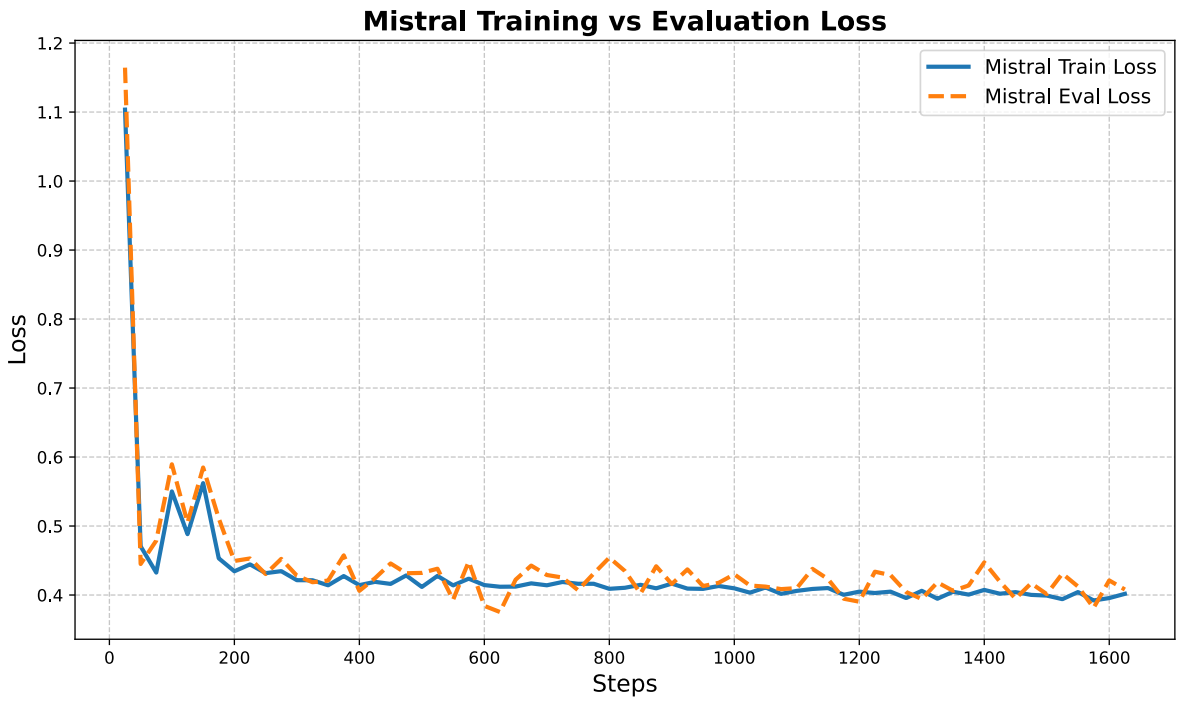


Figure 4.3 Training vs evaluation loss for the Mistral model

### Llama2-7B-chat-hf Loss Analysis (Figure 4.1)

The Llama2 model's loss curves exhibit a typical pattern of rapid loss reduction during the initial stages of training. Both the training and evaluation loss decline sharply within the first 200 steps which indicates that the model quickly adjusts its weights to the given task. After this point, the training loss stabilizes around 0.4 and remains relatively constant throughout the remainder of the fine-tuning process.

The evaluation loss follows a similar trend but shows more fluctuation compared to the training loss. This variability is common when assessing model performance on unseen data and reflects the model's ability to generalize. Despite these fluctuations, the evaluation loss does not deviate significantly from the training loss, suggesting that overfitting is minimal and that the model maintains a reasonable balance between underfitting and overfitting. The low final loss values indicate that the model is learning effectively.

### Llama3-8B-Instruct Loss Analysis (Figure 4.2)

The Llama3 model displays a similar loss reduction pattern to Llama2, with both training and evaluation losses dropping sharply during the first 200 steps. The training loss stabilizes around 0.4 and remains steady throughout the fine-tuning process. However, the evaluation loss exhibits slightly more variability than the Llama2 model, which may suggest that the model's generalization performance is less stable on unseen data.

Nonetheless, the overall evaluation loss remains close to the training loss, indicating that

the model is not overfitting significantly. The fluctuations in the evaluation loss may be attributed to the increased complexity of the Llama3 model.

**Mistral Loss Analysis (Figure 4.3)**

The Mistral model's loss curves present a slightly different pattern from the Llama models, particularly in the early stages of training. Both the training and evaluation loss exhibit some oscillation during the first 200 steps, indicating a higher degree of variance in the learning process during these initial stages. However, after this initial phase, the loss curves stabilize, with both training and evaluation losses converging around a value of 0.4.

The close alignment between training and evaluation loss after 200 steps suggests that the Mistral model generalizes effectively to unseen data. The relatively smooth loss curves after this point reflect a stable learning process, with minimal overfitting. The Mistral model's ability to maintain a low variance in both training and evaluation loss beyond the initial fluctuations indicates that it is well-tuned for the task at hand and is capable of consistent performance across datasets.

**Comparative Analysis**

When comparing the three models, the Llama2 and Llama3 models exhibit relatively stable loss curves, with low variability in the training loss and moderate fluctuations in the evaluation loss. In contrast, the Mistral model, despite showing more variance in the early stages, achieves a closer alignment between training and evaluation loss in the later stages of training. This suggests a better generalization capability.

Overall, the Mistral model demonstrates greater stability after the initial fluctuations which makes it a strong candidate for fine-tuning tasks that require consistent generalization across datasets. While the Llama2 and Llama3 models also perform well, the Llama3 model shows slightly more variability in its evaluation loss, indicating potential areas for further fine-tuning to improve generalization performance.

In summary, these loss curves highlight the importance of stability and alignment between training and evaluation loss as key indicators of a model's performance and generalization capabilities. The Mistral model, in particular, stands out for its strong performance in these areas, making it a promising choice for tasks requiring balanced performance across both training and evaluation datasets.

**LLM Train and Inference Hyperparameters**

| Parameter | Parameter Type | Value |
|---|---|---|
| LoRA $r$ | Train | 256 |
| LoRA Alpha | Train | 128 |
| LoRA Dropout | Train | 0.1 |
| Train Batch Size Per Device | Train | 2 |
| Optimizer | Train | Adam |
| Learning Rate | Train | 3e-4 |
| bf16 | Train | True |
| Number of Training Epochs | Train | 2 |
| Warmup Ratio | Train | 0.05 |
| Learning Rate Scheduler Type | Train | cosine |
| Maximum New Tokens | Inference | 25 |
| Temperature | Inference | 0.7 |
| Quantization | Model | 4-bit |

Table 4.1 Fine-tuning Hyperparameters in Training and Inference Phase

The selected hyperparameters, which are shown in 4.1, for this model fine-tuning and inference process are critical to ensuring the model performs optimally and efficiently. For a meaningful comparison all of the three fine-tuned models were trained with these hyperparameter values. Starting with LoRA $r$, set to 256, defines the rank of the low-rank decomposition used in LoRA layers. Low-Rank Adaptation (LoRA) introduces additional trainable low-rank matrices, allowing for parameter-efficient fine-tuning while retaining high performance [51]. LoRA Alpha, set to 128, is a scaling factor applied to these low-rank matrices. This helps in controlling the balance between fine-tuning and retaining the original model's knowledge.

The LoRA Dropout rate, set to 0.1, provides regularization by randomly deactivating neurons during training, helping reduce overfitting [61]. The Train Batch Size Per Device, specified as 2, defines the number of samples processed in each forward and backward pass per device. A smaller batch size, like 2, trades off between memory usage and the stability of gradient updates [62].

The optimizer chosen is Adam (Adaptive Moment Estimation), known for its adaptive learning rate mechanism, making it ideal for deep learning tasks [63]. The learning rate, set to 3e-4, controls the size of the update steps in gradient descent. A lower learning rate like this helps ensure stable training and prevents overshooting the minima during

optimization.

The bf16, or bfloat16, precision is enabled in this training, which improves memory efficiency without a significant loss in computational precision [64]. This is especially beneficial when working with large models. The number of training epochs is set to 2, meaning the model will pass over the dataset twice. This is typically sufficient for fine-tuning already pre-trained models because LLMs are few-shot learners [65].

A warmup ratio of 0.05 is applied, meaning 5% of the total training steps gradually increase the learning rate from zero, preventing large, unstable gradient updates at the start of training [66]. The learning rate scheduler type is cosine, which smoothly reduces the learning rate following a cosine function [67].

For inference, the maximum number of new tokens is limited to 25. This parameter controls the length of text generated during model inference, limiting output verbosity and maintaining relevance. The temperature, set to 0.7, controls the randomness of the model's predictions. This value balances between deterministic outputs and creative variability, as lower values would make the outputs more focused and deterministic [68].

Finally, the quantization method used is 4-bit, a technique that reduces the size of model weights while maintaining sufficient model accuracy. This is critical for getting faster inference and lower memory consumption [69].

**Tokenizer, Padding and Special Tokens For Instruction**

The tokenizer employed in this study plays a crucial role in how text data is processed for both model training and inference. The tokenizer, derived from the `LlamaTokenizer` class which is a BPE model based on sentencepiece developed by Google, ensures that text is broken down into manageable units (tokens) that the model can process efficiently. Tokenization is particularly important when working with large-scale language models like Mistral, as it directly affects the model's ability to understand and generate coherent outputs. The tokenizer's configuration includes specific special tokens, such as `<unk>`, `<s>`, and `</s>`, which represent unknown tokens, the start of a sequence, and the end of a sequence, respectively. These tokens guide the model in structuring input-output sequences and handling edge cases in natural language processing (e.g., handling out-of-vocabulary words) [70].

An important aspect of tokenization is padding, which ensures that all input sequences have uniform length, especially when processing data in batches. In this configuration, padding is performed on the right side, meaning that the padded tokens are appended to the end of the sequence, allowing the model to process shorter sequences without introducing noise at the start. The padding token used in this case is the end-of-sequence

| Model | Hyperparameter | Value |
|---|---|---|
| LSTM | Hidden Layer Size | 128 |
| LSTM/CNN | Batch Size | 64 |
| | Optimizer | Adam |
| | Learning Rate | 0.001 |
| | Epochs | 40 |
| CNN | Convolutional Layers | 2 |
| | Channels in Conv1 | 32 |
| | Channels in Conv2 | 64 |
| | Global Pooling Type | Adaptive Average Pooling |

Table 4.2 Hyperparameters used for training the LSTM and CNN baseline models

token (`<eos>`), as the tokenizer does not include a dedicated padding token. This method simplifies the padding process while maintaining semantic consistency, as the end-of-sequence marker fits logically as a padding token [45]. Right-side padding is commonly used in transformers due to its ability to preserve the earlier, more informative parts of a sequence at the beginning of the input [71].

Moreover, the instruction format used in fine-tuning enhances the tokenizer's role by introducing the `[INST]` and `[/INST]` tokens, which encapsulate specific prompts or tasks for the model. These tokens create a clear structure that separates user input from the Mistral model's responses. By including these instruction tokens, the model can more effectively align its generation with instruction-following behavior, which is crucial in applications such as conversational agents or task-specific generators. This format also facilitates the efficient switching of roles between user and assistant, promoting clearer interactions [39].

Overall, the choices made in tokenizer configuration, padding strategy, and the use of instruction format tokens significantly enhance the model's performance in instruction-following tasks, ensuring robust and efficient text processing during both training and inference stages.

### 4.2.2 Baseline Hyperparameters

The hyperparameters, shown in 4.2, used in both the LSTM and CNN models play a crucial role in defining the behavior and performance of the models during training.

In the LSTM model, the input size represents the dimensionality of each input in the sequence, which in this case is set to 1. This corresponds to the encoded transaction category for each transaction made by a customer. The hidden layer size controls the model's capacity to capture temporal dependencies. With 128 hidden units, the model can balance between complexity and computational efficiency, allowing it to model complex relationships in sequential data.

For both the LSTM and CNN models, the batch size determines how many samples are processed before updating the model's weights. A batch size of 64 strikes a balance between memory usage and training efficiency. The optimizer, Adam, is a well-regarded optimization algorithm that combines momentum with adaptive learning rates, making it effective for training deep learning models. Adam's learning rate is set to 0.001, a typical value that allows the model to converge at a controlled pace without making drastic updates to the weights during each iteration. The models are trained over 40 epochs, giving them enough time to learn meaningful patterns from the data without risking overfitting.

In the CNN model, additional hyperparameters such as the number of convolutional layers and channels define the model's ability to learn local patterns from the input sequences. The CNN architecture includes two convolutional layers. The first layer has 32 filters, while the second has 64, allowing the model to progressively learn more complex and abstract features from the data. The use of global pooling, specifically adaptive average pooling, helps reduce the dimensionality of the learned feature maps by summarizing the features into a compact form, which is useful for reducing computational cost and preventing overfitting.

These hyperparameters together define the behavior of the models during training and how well they can generalize to new data. The combination of parameters like the learning rate, batch size, and optimizer ensures stable and efficient training, while the architecture-specific parameters like hidden layer size for LSTM and convolutional filters for CNN define how well each model can capture and learn patterns in the sequential transaction data.

Anohter baseline method was based on the logic of averaging. This simplest model used for benchmarking is based on the historical average frequencies of transaction categories for each client. The prediction of a purchase in a specific category by a client is calculated using the formula:

$$p_{ij} = \frac{1}{K_i} \sum_{k=1}^{K_i} I(n_{ijk} > 0) \tag{4.6}$$

Here, $p_{ij}$ represents the probability of the $i$-th client making a purchase in the $j$-th category, $K_i$ denotes the total number of time periods considered, $n_{ijk}$ is the number of transactions for the $i$-th client in the $j$-th category at the $k$-th time period, and $I$ is an indicator function that evaluates to 1 if the client has made one or more transactions in that category during the period.

The second model that has been trained for performance comparison was a simple LSTM architecture, with vectors representing encoded transaction categories serving as input. This LSTM is configured with a hidden state dimensionality of 128 and is trained to capture temporal transaction patterns using Backpropagation Through Time (BPTT) and a cross-entropy loss function suited for classification tasks.

Lastly, a Convolutional Neural Network (CNN) model was trained. The network's input layer accepts concatenated vectors of encoded transaction categories, feeding into a straightforward CNN model with two 2D convolutional layers followed by a pooling layer. The network's architecture is completed with a final output layer that shares the same loss function as the LSTM, enabling direct performance comparisons between the recurrent and convolutional network strategies.

### 4.2.3 Varying Sequence Lenghts

An additional critical aspect of the evaluation process involved sourcing predictions from the models using diverse input sequence lengths. To elaborate, the models were originally trained using the last nine transactions from a customer's history, with the tenth transaction serving as a target for prediction because this sequence length has proven itself to be more successful in various experiments (the detailed experiment results table available in the appendix Table 1). In the testing phase, however, input sequences were altered to include only the four most recent transactions, the last seven, or an extended set of fourteen transactions. For illustrative purposes, consider the process of inputting a customer's demographic information into the model, along with their transaction history. Instead of the full sequence the model was trained on (last nine), it received a truncated or extended series of transactions. This meant that the trained models underwent evaluation with a varying quantity of transactional data.

The very first reason for employing various sequence lengths in the testing phase, despite all models being trained on a fixed sequence length, is to rigorously assess their generalizability. This step is critical because it allows researchers to understand how well a model trained on a particular pattern of data, such as the last nine transactions from customers of Bank A, can adapt to and accurately predict outcomes when presented with different patterns. Generalizability is a cornerstone of model efficacy in real-world applications where data patterns are unlikely to remain consistent. For instance, in practical scenarios, models may encounter customers with fewer than nine transactions. If the models were only trained and tested on the last nine transactions, their performance might overestimate their real-world effectiveness. By introducing variability in the input sequence lengths, one can simulate a more realistic environment where data availability fluctuates.

Furthermore, the investigation into robustness against sequence length variability goes hand in hand with the need for generalizability. The premise here is to explore the stability of the models' performance when the volume of input data deviates from the training conditions. Should a model exhibit comparable accuracy across various input lengths, it would be deemed robust, indicating reliability and utility in real-world scenarios where data sparsity or abundance can occur.

Lastly, the purpose of deriving multiple scores from tests using different sequence lengths is to delineate the performance boundaries of the models. Instead of a singular score that provides a limited view, several scores sketch a fuller picture of each model's performance landscape. This comprehensive assessment is akin to a multi-faceted examination of the models. It allows for a more nuanced understanding of where a model's predictions can be trusted and where caution should be exercised. This holistic evaluation is particularly relevant in machine learning applications in finance, where an incomplete understanding of a model's capabilities can lead to misguided trust and potentially costly errors in transaction category predictions. By determining the success bounds through varied sequence lengths, one ensures that the models are not just theoretically sound but practically versatile and reliable.

| Model | Dataset | Sequence Length | Accuracy | Precision | Recall | F1 (weighted) |
|---|---|---|---|---|---|---|
| Randomized Prediction | Bank B | NA | 0.24 | 0.47 | 0.24 | 0.29 |
| Proportional/Weighted Prediction | | NA | 0.39 | 0.49 | 0.39 | 0.43 |
| Mistral Instruct 7b v.2 (Raw Model) | | last-9 | 0.51 | 0.50 | 0.50 | 0.49 |
| Baseline (Averaging) | | last-4 | 0.43 | 0.52 | 0.43 | 0.43 |
| | | last-7 | 0.42 | 0.50 | 0.42 | 0.42 |
| | | last-9 | 0.42 | 0.50 | 0.42 | 0.43 |
| | | last-14 | 0.43 | 0.49 | 0.43 | 0.44 |
| Hidden Markov Chain Modeling | | last-4 | 0.52 | 0.52 | 0.51 | 0.48 |
| | | last-7 | 0.52 | 0.51 | 0.52 | 0.50 |
| | | last-9 | 0.51 | 0.49 | 0.51 | 0.50 |
| | | last-14 | 0.49 | 0.51 | 0.49 | 0.44 |
| CNN (trained on Bank A) | | last-4 | 0.63 | 0.60 | 0.63 | 0.60 |
| | | last-7 | 0.64 | 0.61 | 0.64 | 0.61 |
| | | last-9 | 0.65 | 0.61 | 0.65 | 0.62 |
| | | last-14 | 0.66 | 0.62 | 0.66 | 0.62 |
| LSTM (trained on Bank A) | | last-4 | **0.63** | 0.59 | **0.62** | 0.58 |
| | | last-7 | **0.64** | 0.61 | **0.64** | 0.60 |
| | | last-9 | 0.62 | 0.60 | 0.62 | 0.60 |
| | | last-14 | **0.63** | 0.62 | **0.63** | 0.61 |
| LLaMA-2-Chat 7B (trained on Bank A) | | last-4 | 0.53 | 0.61 | 0.52 | 0.54 |
| | | last-7 | 0.52 | 0.56 | 0.54 | 0.55 |
| | | last-9 | 0.58 | 0.59 | 0.58 | 0.58 |
| | | last-14 | 0.54 | 0.58 | 0.54 | 0.55 |
| LLaMA-3-8B (trained on Bank A) | | last-4 | 0.56 | 0.64 | 0.55 | 0.57 |
| | | last-7 | 0.55 | 0.59 | 0.57 | 0.58 |
| | | last-9 | 0.61 | 0.62 | 0.61 | 0.61 |
| | | last-14 | 0.57 | 0.61 | 0.57 | 0.58 |
| Mistral Instruct 7b v.2 (trained on Bank A) | | last-4 | 0.61 | **0.69** | 0.60 | **0.62** |
| | | last-7 | 0.60 | **0.64** | 0.62 | **0.63** |
| | | last-9 | **0.66** | **0.67** | **0.66** | **0.66** |
| | | last-14 | 0.62 | **0.66** | 0.62 | **0.63** |

Table 4.3 Overall Results for All the Experimented Models

| Model | Dataset | Sequence Length | Clothing | Gas Stations | Grocery | Other |
|---|---|---|---|---|---|---|
| Randomized Prediction | Bank B | NA | 0.04 | 0.13 | 0.25 | 0.34 |
| Proportional/Weighted Prediction | | NA | 0.09 | 0.14 | 0.29 | 0.54 |
| Hidden Markov Chain Modeling | | last-4 | 0.01 | 0.30 | 0.51 | 0.60 |
| | | last-7 | 0.07 | 0.34 | 0.47 | 0.61 |
| | | last-9 | 0.01 | 0.32 | 0.46 | 0.58 |
| | | last-14 | 0.02 | 0.19 | 0.32 | 0.55 |
| CNN | | last-4 | 0.11 | 0.19 | 0.42 | **0.75** |
| | | last-7 | 0.02 | 0.21 | 0.42 | **0.76** |
| | | last-9 | 0.00 | 0.22 | 0.42 | **0.76** |
| | | last-14 | 0.00 | 0.22 | 0.41 | **0.75** |
| LSTM | | last-4 | 0.01 | 0.20 | 0.35 | **0.76** |
| | | last-7 | 0.00 | 0.22 | 0.45 | **0.74** |
| | | last-9 | 0.00 | 0.23 | 0.45 | **0.75** |
| | | last-14 | 0.00 | 0.24 | 0.47 | **0.74** |
| LLaMA-2-Chat 7B | | last-4 | **0.40** | **0.39** | **0.48** | 0.60 |
| | | last-7 | **0.04** | **0.42** | **0.44** | 0.62 |
| | | last-9 | **0.54** | **0.32** | **0.51** | 0.67 |
| | | last-14 | **0.14** | **0.40** | **0.47** | 0.63 |
| LLaMA-3-8B | | last-4 | **0.43** | **0.42** | **0.51** | 0.63 |
| | | last-7 | **0.07** | **0.45** | **0.47** | 0.65 |
| | | last-9 | **0.57** | **0.35** | **0.54** | 0.70 |
| | | last-14 | **0.17** | **0.43** | **0.50** | 0.66 |
| Mistral Instruct 7b v.2 | | last-4 | **0.48** | **0.47** | **0.56** | 0.68 |
| | | last-7 | **0.12** | **0.50** | **0.52** | 0.70 |
| | | last-9 | **0.62** | **0.40** | **0.59** | 0.75 |
| | | last-14 | **0.22** | **0.48** | **0.55** | 0.71 |

Table 4.4 Class-wise F1 scores for all experimented models

## 4.3 Results

The results of the experimental evaluation are organized to illustrate the performance of various models across a series of sequence lengths and to show the ability of each model to predict user preferences for merchant categories in a multiclass classification framework. The dataset utilized for training and validation purposes, Bank A, served as the basis for training several models. Bank B, an entirely separate dataset not used for training, provided an additional layer of testing to assess the models' generalization capabilities.

Table 1 shows the results of baseline, neural networks (CNN and LSTM), and the fine-tuned LLM model. The bold scores were depicted by comparing LSTM and CNN with Mistral model. The "Dataset" column indicates the specific dataset used for making predictions or inferences with each model. Sequence length values demonstrate how many transactions of customers were used as input for the models to get a transaction category prediction. For instance, "last-7" means that the last seven transactions of customers were given as input for a model to get an inference regarding their 8th transaction category. It's important to note that all models, with the exception of the baseline averaging model which does not require any training, were trained using the last nine transactions ("last-9") from customers in Bank A dataset and the 10th transaction category used as a ground truth or label.

### 4.3.1 Overall Results in Generalization to Bank B Data

Non-trained baseline models (like Randomized Prediction or **Proportional/Weighted** Prediction which has a simplistic strategy by aligning predictions for Bank B with the class distribution observed in Bank A given not applicable sequence length because they are about distributions and not purchases sequences) shows consistently lower performance metrics across accuracy, precision, recall, and F1 scores when compared to neural network approaches and the fine-tuned LLM. Specifically, the model's accuracy hovers around 0.42 to 0.43 across different sequence lengths, with precision and recall metrics similarly ranging between 0.49 to 0.52 and 0.42 to 0.43, respectively. The F1 score, which considers both precision and recall, remains relatively low, ranging from 0.42 to 0.44. The Baseline model's performance, while providing a necessary benchmark for comparison, highlights its inadequacy for complex predictive tasks such as merchant category forecasting. Its lower performance metrics relative to other evaluated models reinforce the

importance of adopting more sophisticated models that can better capture and analyze the intricacies of financial transaction data.

The Hidden Markov Model (HMM) demonstrates a modest improvement over baseline models in overall and class-specific performance. As shown in Table 1, the HMM achieves an accuracy ranging from 0.49 to 0.52, with an F1 score peaking at 0.50 for the last-7 sequence length. This performance is moderately better than randomized or proportional prediction. In Table 4.4, the class-specific F1 scores reveal that HMM handles the 'Grocery' and 'Other' categories better than 'Clothing' and 'Gas Stations,' with the 'Other' category achieving an F1 score of 0.60 at the last-4 sequence length. However, its limited ability to handle minority classes like 'Clothing,' where scores are as low as 0.01, underscores its inadequacy for imbalanced datasets compared to neural networks or fine-tuned models.

The analysis of the Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) models, both trained on Bank A's data and tested on Bank B, demonstrates their relative performance metrics in predicting the next merchant categories. The CNN model exhibits moderate effectiveness across various sequence lengths, with the best performance observed at the last-14 transactions, achieving an accuracy of 0.66, a precision of 0.62, and a recall of 0.66, resulting in an F1 score of 0.62. When considering shorter sequence lengths, the performance slightly decreases, with the model demonstrating accuracy and F1 score around 0.63-0.65 and 0.60-0.62 respectively. These metrics suggest that while the CNN can handle different contexts, its predictive capabilities are not strongly enhanced by extending the sequence length.

The LSTM model, known for its ability to capture temporal dependencies, shows a consistent pattern across different transaction sequence lengths. The performance peaks with a sequence length of the last-14 transactions, where it achieves an accuracy of 0.63, a precision of 0.62, and a recall of 0.63, yielding an F1 score of 0.61. Similar to the CNN, the LSTM model does not demonstrate substantial improvement as the sequence length increases, with metrics revolving around 0.62 for accuracy and 0.60 for the F1 score across different contexts.

The fine-tuned Mistral Instruct model achieves a weighted F1 score of 0.66 when evaluating the last-9 transactions, which is notably higher than the scores obtained by both the CNN and LSTM models under the same conditions (0.62 and 0.60 respectively). These superior F1 scores across all sequence lengths indicate that the fine-tuned Mistral Instruct model not only predicts more accurately but also maintains a balanced sensitivity and precision, which is crucial in handling class-imbalanced datasets effectively. When compared to the other baseline models, the fine-tuned Mistral Instruct stands out, particularly in terms of the F1 score, which is crucial for evaluating performance in an

imbalanced dataset context. The model not only outperforms the CNN and LSTM in this metric but also exhibits a higher consistency across different sequence lengths. This highlights its adaptability and overall superior predictive power in this specific task.

### 4.3.2 Class-Specific Performance

In Table 4.4, a detailed analysis of the classification performance across individual merchant categories for three models that are all trained on Bank A dataset is presented. The distribution of classes in the training set from Bank A (also similar to Bank B data) was notably skewed, with the grocery category at 31.3%, 11.9% to gas stations, and 11.2% to clothing. The remaining 45.5% of transactions pertain to categories different than grocery, clothing, and gas stations. This class imbalance presents a challenge for predictive models, particularly for minority classes that are underrepresented in the data.

The evaluation of class-specific F1 scores offers a comprehensive view of the performance disparities between the fine-tuned Mistral Instruct 7b v.2 model and traditional sequential models such as CNNs and LSTMs. This analysis is critical as it illustrates the refined capability of the Mistral model to handle minority classes with a higher level of accuracy and efficiency, compared to the more generic treatment of classes by the other models.

In the 'Clothing' category, the Mistral Instruct model demonstrates a pronounced superiority with an F1 score of 0.620 when analyzing the last-9 transaction sequences. This score is significantly higher than those achieved by the CNN and LSTM models, which reach only 0.107 and 0.006, respectively. Such a stark difference underscores the fine-tuned model's approach to semantic learning and prediction in categories that represent a smaller portion of the dataset, specifically the 11.2% comprising clothing transactions in Bank A's data.

Similarly, for the "Gas Stations" category, the Mistral model achieves its peak performance with an F1 score of 0.500 using the last-7 transactions. This outstrips the CNN and LSTM models, which max out at F1 scores of 0.222 and 0.244. The improved performance in this category, which constitutes 11.9% of the transactions in the training dataset, highlights the model's effective adaptation to categories characterized by lower frequency yet distinct spending patterns.

The "Grocery" category reveals the model's exceptional ability to manage relatively more frequent transaction classes but still not the majority, with an impressive F1 score of 0.59 from the last-9 transaction sequences. This compares favorably with the best scores

from CNN and LSTM, which are 0.422 and 0.474, respectively. Given that grocery transactions make up 31.3% of Bank A's data, this result emphasizes the Mistral model's robust predictive capabilities to handle dominant classes effectively as well.

These findings not only reinforce the effectiveness of the Mistral model in dealing with class imbalances but also highlight its suitability for complex predictive tasks in real-world financial environments. The model's ability to excel in predicting minority class categories, where traditional models struggle, sets a new standard for the usage of LLMs in the domain of financial transactions. The detailed class-specific performance thus underlines the transformative impact of fine-tuning large language models on domain-specific tasks.

The fine-tuned Mistral Instruct 7b v.2 model exhibits remarkable consistency across all categories. This demonstrates its stability and reliability in performance when compared to traditional sequential models like CNNs and LSTMs. This consistency is not only evident in the superior scores it achieves in the minority categories but also in its robust performance across varying transaction sequences and class distributions. Unlike the CNN and LSTM models, which show considerable fluctuations in their class-specific F1 scores, the Mistral model maintains a more uniform performance spectrum. For instance, while the F1 scores for CNN and LSTM vary widely from nearly zero in some categories to higher values in others, the Mistral model's scores remain notably higher and more stable. This shows more stability in reducing the performance variance between the best and worst-performing categories. This consistent performance is crucial for applications that require dependable and predictable model behavior across diverse and potentially imbalanced datasets.

# 5. Conclusion

The primary motivation of this study was to explore the potential of LLMs in the domain of financial transaction prediction, particularly focusing on the challenging task of predicting the next purchase merchant category based on historical transaction data. Traditional deep learning models such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks have shown varying degrees of success in this domain but often fall short in capturing the contexts inherent in transaction data. This work aimed to bridge this gap by leveraging the advanced capabilities of fine-tuned LLMs, demonstrating their superior performance and broader applicability.

The key contribution of this study lies in the innovative application of the Mistral 7B Instruct model to predict the next purchase merchant categories. The model demonstrated an enhanced ability to handle the complexities and imbalances of financial transaction data more effectively than traditional models. By fine-tuning the LLM on transactional tabular data reformatted into personalized instructions, model's predictive accuracy has been significantly enhanced. This approach not only highlights the versatility of LLMs in understanding and modeling human behavior but also sets a new benchmark for predictive modeling in financial contexts.

**Summary of Work**

In this study, a novel methodology is proposed for predicting next merchant categories from financial transaction data by fine-tuning an open-source Large Language Model (LLM), specifically the Mistral 7B Instruct model. The fine-tuning process leverages the Low-Rank Adaptation (LoRA) technique, which is shown to enable efficient and domain-specific adaptation of the pre-trained model without sacrificing its inherent knowledge. The unique contribution of this work lies in the reformatting of transactional tabular data into natural language instructions, allowing the LLM to process and learn from customer demographic and transactional histories as narrative inputs. This approach provides the model with a more contextual understanding of the data, as opposed to traditional deep learning models like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, which often struggle with capturing the semantic nuances in

financial data.

The fine-tuned LLM is shown to outperform the baseline and deep learning models across a range of evaluation metrics, particularly in terms of accuracy, precision, recall, and weighted F1 scores. The performance of the Mistral 7B model is demonstrated through a rigorous experimental evaluation, utilizing datasets from two distinct banks—Bank A for training and Bank B for testing. The cross-bank testing, which assesses the model's generalization capabilities, is shown to confirm that the fine-tuned LLM maintains robust predictive power even on unseen data. Moreover, the model is tested across varying input sequence lengths to simulate real-world variability in customer transaction histories. This fine-tuned LLM consistently achieves superior results, particularly in key merchant categories such as grocery, clothing, and gas stations, where traditional models often underperform, especially when dealing with imbalanced data.

A key advantage of the fine-tuned Mistral model is its ability to effectively manage class imbalances, a frequent issue in real-world financial transaction data where certain categories are underrepresented. The fine-tuned LLM is shown to handle minority categories, such as clothing and gas stations, with far greater precision and recall than CNN and LSTM models, making it particularly valuable for applications that require balanced performance across a wide range of transaction categories. In contrast to conventional models, which often overfit to majority classes, the LLM demonstrates a more nuanced understanding of both majority and minority classes, significantly improving its performance in predicting the next merchant category in a sequence.

The findings from this study highlight the transformative potential of LLMs in modeling complex human behaviors in financial contexts, particularly in tasks that involve sequential prediction. By fine-tuning the Mistral model on domain-specific data, the study is shown to offer a powerful and flexible alternative to traditional deep learning models, requiring less manual feature engineering while achieving higher accuracy. This research not only advances the state-of-the-art in predictive modeling within the financial sector but also illustrates how LLMs can be applied to human behavior modeling tasks more broadly.

**Future Work**

Building upon the promising results demonstrated in this study, several avenues for future research can be explored to further enhance the use of fine-tuned Large Language Models (LLMs) in financial transaction prediction and human behavior modeling. The current landscape of LLMs for recommendation systems and behavior modeling is still evolving, and there are substantial opportunities to extend this work by addressing limitations, exploring new techniques, and applying these models to broader contexts.

One potential area of improvement is the incorporation of multi-modal data into LLMs for transaction prediction tasks. As outlined in the literature, various studies have successfully integrated visual, textual, and tabular data to enhance the performance of recommendation systems. For example, multi-view recurrent neural networks (MV-RNN) combine visual and textual data for more comprehensive user modeling, and models like GPTRec have explored sequential recommendations through sophisticated embedding techniques. Extending the current approach by incorporating multi-modal data, such as customer interaction logs, geographic information, or even social media activity, could further enrich the model's understanding of user preferences. This integration could be particularly useful in financial contexts because customer behavior is often influenced by external factors beyond transactional history alone.

Additionally, further exploration into the use of LLMs in cross-domain or zero-shot learning scenarios would be a valuable extension of this work. The literature on zero-shot recommendation tasks highlights the potential of LLMs to operate effectively without extensive retraining on specific datasets. This capability could be particularly advantageous in financial sectors, where data privacy concerns often limit access to large datasets for training. Investigating the effectiveness of the fine-tuned Mistral model in zero-shot settings or adapting it to cross-domain recommendations could enhance its applicability across different banks or even entirely different industries. Furthermore, frameworks such as Prompt4NR and ZESRec have illustrated the potential of prompt learning and zero-shot strategies in mitigating cold-start problems.

Finally, another promising area for future work involves enhancing the model's interpretability and explainability. As discussed in several studies, the ability to provide transparent and understandable predictions is crucial in sensitive domains like finance, where decision-makers need clear justifications for model outputs. Incorporating explainability techniques, such as personalized prompt learning , into LLM-based systems could allow users to better understand why certain predictions are made, thus improving trust and adoption of these models in real-world financial applications. This could also help address biases that have been observed in LLMs, as noted in various studies where pretrained language models exhibited linguistic biases. By focusing on explainability, future research could contribute to the development of more equitable and accountable predictive models, further enhancing their utility in financial decision-making.

In summary, future research should aim to broaden the scope of LLMs in behavior prediction by incorporating multi-modal data, exploring cross-domain and zero-shot learning, and improving model transparency. These directions will help refine the models and make them more versatile, ultimately enabling their application to more complex, real-world financial and recommendation scenarios.

# BIBLIOGRAPHY


[1] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, et al. A survey on large language models for recommendation, 2023.

[2] Yuhui Zhang, Hao Ding, Zeren Shui, Yifei Ma, James Zou, Anoop Deoras, and Hao Wang. Language models as recommender systems: Evaluations and limitations. In *NeurIPS 2021 Workshop on I (Still) Can't Believe It's Not Better*, 2021.

[3] Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. Chat-rec: Towards interactive and explainable llms-augmented recommender system, 2023.

[4] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt and predict paradigm (p5), 2023.

[5] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. Tallrec: An effective and efficient tuning framework to align large language models with recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*. ACM, Sep 2023.

[6] Yupeng Hou, Zhankui He, Julian McAuley, and Wayne Xin Zhao. Learning vector-quantized item representation for transferable sequential recommenders, 2023.

[7] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation, 2018.

[8] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023.

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, et al. Palm: Scaling language modeling with pathways, 2022.

[10] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, et al. Holistic evaluation of language models, 2023.

[11] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, et al. Emergent abilities of large language models, 2022.

[12] Christopher A. Bail. Can generative ai improve social science? *Proceedings of the National Academy of Sciences*, 121(21), May 2024.

[13] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, et al. Instruction tuning for large language models: A survey, 2024.

[14] Jiahao Wang, Bolin Zhang, Qianlong Du, Jiajun Zhang, and Dianhui Chu. A survey on data selection for llm instruction tuning, 2024.

[15] Egor Shikov and Klavdiya Bochenina. Forecasting purchase categories by transactional data: A comparative study of classification methods. In *Lecture Notes in Computer Science*, Lecture Notes in Computer Science, pages 249–262. Springer International Publishing, Cham, 2019.

[16] Aleksandr V. Petrov and Craig Macdonald. Generative sequential recommendation with gptrec, 2023.

[17] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer, 2019.

[18] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. Recommendation as instruction following: A large language model empowered recommendation approach, 2023.

[19] Zeyu Cui, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. M6-rec: Generative pretrained language models are open-ended recommender systems, 2022.

[20] Lei Wang and Ee-Peng Lim. Zero-shot next-item recommendation using large pretrained language models, 2023.

[21] Zizhuo Zhang and Bang Wang. Prompt learning for news recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23. ACM, July 2023.

[22] Lei Li, Yongfeng Zhang, and Li Chen. Personalized prompt learning for explainable recommendation, 2023.

[23] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Towards universal sequence representation learning for recommender systems, 2022.

[24] Fan Yang, Zheng Chen, Ziyan Jiang, Eunah Cho, Xiaojiang Huang, and Yanbin Lu. Palr: Personalization aware llms for recommendation, 2023.

[25] Damien Sileo, Wout Vossen, and Robbe Raymaekers. Zero-shot recommendation as language modeling, 2021.

[26] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. Large language models are zero-shot rankers for recommender systems, 2024.

[27] Hao Ding, Yifei Ma, Anoop Deoras, Yuyang Wang, and Hao Wang. Zero-shot recommender systems, 2021.

[28] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation, 2016.

[29] Tariq Mahmood and Francesco Ricci. Learning and adaptivity in interactive recommender systems. In *Proceedings of the Ninth International Conference on Electronic Commerce*, pages 75–84, New York, NY, USA, 2007. Association for Computing Machinery.

[30] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 811–820. Association for Computing Machinery, 2010.

[31] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations, 2020.

[32] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Orgun. Sequential recommender systems: Challenges, progress and prospects. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Aug 2019.

[33] Qiang Cui, Shu Wu, Qiang Liu, Wen Zhong, and Liang Wang. Mv-rnn: A multi-view recurrent neural network for sequential recommendation, 2018.

[34] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 152–160. Association for Computing Machinery, 2017.

[35] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2016.

[36] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding, 2018.

[37] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation, 2018.

[38] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation, 2016.

[39] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[40] Shoujin Wang, Quan Z Sheng, Tao Liu, Jinpeng Yu, Abbas Sadeghi-Niaraki, and Wei Emma Zhang. Sequential recommender systems: challenges, progress and prospects. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6332–6338, 2019.

[41] Liyilei Su, Xumin Zuo, Rui Li, Xin Wang, Heng Zhao, and Bingding Huang. A systematic review for transformer-based long-term series forecasting, 2023.

[42] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[43] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, et al. Stanford alpaca: An instruction-following llama model. GitHub repository, 2023.

[44] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, et al. Mistral 7b, 2023.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[46] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

[47] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebr'on, and Sumit K. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *ArXiv*, abs/2305.13245, 2023.

[48] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.

[49] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

[50] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, and Nathan Lambert. Trl: Transformer reinforcement learning.

[51] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[52] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

[53] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics.

[54] Erdem Kaya, Xiaowen Dong, Yoshihiko Suhara, Selim Balcisoy, Burcin Bozkaya, and Alex Pentland. Behavioral attributes and financial churn prediction. *EPJ Data Science*, 7, October 2018.

[55] Vivek Kumar Singh, Burcin Bozkaya, and Alex Pentland. Money walks: Implicit mobility behavior and financial well-being. *PLOS ONE*, 10(8), August 2015.

[56] Mistral AI Team. Announcing mistral 7b, September 27 2023. Accessed: 2024-10-20.

[57] HuggingFace. Peft, September 27 2023. Accessed: 2024-10-20.

[58] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[59] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, et al. Llama: Open and efficient foundation language models, 2023.

[60] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, et al. Self-instruct: Aligning language models with self-generated instructions, 2023.

[61] Nitish Srivastava, Geoffrey Hinton, et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[62] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.

[63] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[64] Daya Kalamkar et al. A study of bfloat16 for deep learning training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–9, 2019.

[65] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[66] Martin Popel and Ondřej Bojar. Training tips for the transformer model. In *Proceedings of the third conference on machine translation: Research papers*, pages 43–52, 2018.

[67] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[68] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020.

[69] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021.

[70] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

[71] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*, 2018.

| Model | Train Dataset | Test Dataset | Train Sequence Length | Test Sequence Length | Clothing | Gas Stations | Grocery | Other |
|---|---|---|---|---|---|---|---|---|
| LSTM | Bank A | Bank B | last-4 | last-4 | 0.12 | 0.39 | 0.300 | **0.75** |
| | | | | last-7 | 0.09 | 0.38 | 0.410 | **0.75** |
| | | | | last-9 | 0.08 | 0.38 | 0.420 | **0.77** |
| | | | | last-14 | 0.06 | 0.36 | 0.390 | **0.75** |
| | | | last-7 | last-4 | 0.06 | 0.36 | 0.360 | **0.72** |
| | | | | last-7 | 0.05 | 0.35 | 0.400 | **0.74** |
| | | | | last-9 | 0.04 | 0.35 | 0.430 | **0.75** |
| | | | | last-14 | 0.03 | 0.35 | 0.460 | **0.77** |
| | | | last-9 | last-4 | 0.006 | 0.200 | 0.346 | **0.763** |
| | | | | last-7 | 0.003 | 0.223 | 0.448 | **0.744** |
| | | | | last-9 | 0.001 | 0.230 | 0.450 | **0.750** |
| | | | | last-14 | 0.000 | 0.244 | 0.474 | **0.740** |
| | | | last-14 | last-4 | 0.11 | 0.38 | 0.346 | **0.72** |
| | | | | last-7 | 0.06 | 0.37 | 0.420 | **0.76** |
| | | | | last-9 | 0.06 | 0.38 | 0.410 | **0.76** |
| | | | | last-14 | 0.11 | 0.57 | 0.430 | **0.73** |
| Mistral | Bank A | Bank B | last-4 | last-4 | **0.330** | **0.400** | **0.520** | 0.630 |
| | | | | last-7 | **0.100** | **0.450** | **0.480** | 0.610 |
| | | | | last-9 | **0.350** | **0.340** | **0.550** | 0.630 |
| | | | | last-14 | **0.220** | **0.420** | **0.500** | 0.650 |
| | | | last-7 | last-4 | **0.450** | **0.400** | **0.510** | 0.680 |
| | | | | last-7 | **0.090** | **0.550** | **0.460** | 0.640 |
| | | | | last-9 | **0.310** | **0.430** | **0.530** | 0.610 |
| | | | | last-14 | **0.190** | **0.480** | **0.480** | 0.630 |
| | | | last-9 | last-4 | **0.480** | **0.470** | **0.560** | 0.680 |
| | | | | last-7 | **0.120** | **0.500** | **0.520** | 0.700 |
| | | | | last-9 | **0.620** | **0.400** | **0.590** | 0.750 |
| | | | | last-14 | **0.220** | **0.480** | **0.550** | 0.710 |
| | | | last-14 | last-4 | **0.380** | **0.450** | **0.550** | 0.640 |
| | | | | last-7 | **0.010** | **0.440** | **0.510** | 0.710 |
| | | | | last-9 | **0.150** | **0.350** | **0.550** | 0.700 |
| | | | | last-14 | **0.230** | **0.450** | **0.490** | 0.680 |

Table 1 LSTM and Mistral Model Class-wise F1 Scores In Varying Sequence Length