Better with Less: Small Proprietary Models Surpass Large Language Models in Financial Transaction Understanding

Wanying Ding
JPMorgan Chase & Co.
Palo Alto, California, USA
wanying.ding@jpmchase.com

Savinay Narendra
JPMorgan Chase & Co.
Palo Alto, California, USA
savinay.narendra@jpmchase.com

Xiran Shi
JPMorgan Chase & Co.
Palo Alto, California, USA
xiran.shi@jpmchase.com

Adwait Ratnaparkhi
JPMorgan Chase & Co.
Palo Alto, California, USA
adwait.ratnaparkhi@jpmchase.com

Chengrui Yang
JPMorgan Chase & Co.
New York, New York, USA
chengrui.yang@chase.com

Nikoo Sabzevar
JPMorgan Chase & Co.
Chicago, Illinois, USA
nikoo.sabzevar@chase.com

Ziyan Yin
JPMorgan Chase & Co.
Wilmington, Delaware, USA
ziyan.yin@chase.com

Abstract

Analyzing financial transactions is crucial for ensuring regulatory compliance, detecting fraud, and supporting decisions. The complexity of financial transaction data necessitates advanced techniques to extract meaningful insights and ensure accurate analysis. Since Transformer-based models have shown outstanding performance across multiple domains, this paper seeks to explore their potential in understanding financial transactions. This paper conducts extensive experiments to evaluate three types of Transformer models: Encoder-Only, Decoder-Only, and Encoder-Decoder models. For each type, we explore three options: pretrained LLMs, fine-tuned LLMs, and small proprietary models developed from scratch. Our analysis reveals that while LLMs, such as LLaMA3-8b, Flan-T5, and SBERT, demonstrate impressive capabilities in various natural language processing tasks, they do not significantly outperform small proprietary models in the specific context of financial transaction understanding. This phenomenon is particularly evident in terms of speed and cost efficiency. Proprietary models, tailored to the unique requirements of transaction data, exhibit faster processing times and lower operational costs, making them more suitable for real-time applications in the financial sector. Our findings highlight the importance of model selection based on domain-specific needs and underscore the potential advantages of customized proprietary models over general-purpose LLMs in specialized applications. Ultimately, we chose to implement a proprietary decoder-only model to handle the complex transactions that we previously couldn't manage. This model can help us to improve 14% transaction coverage, and save more than $13 million annual cost.

Keywords

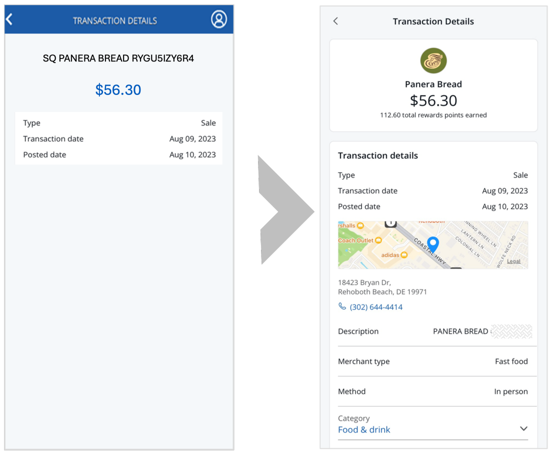
Financial Transaction Understanding, Merchant Retrieval, Transformer Model, Large Language Model, Natural Language Understanding

1 Introduction

In the payments industry, the ability to accurately understand transactions is crucial for assessing business risks, enhancing user experiences, and minimizing inquiry costs. However, this task is complicated by the fact that transaction data is often messy and expressed in various formats. Traditional rule-based methods can manage data from a limited number of merchants, but they struggle to handle the vast array of unseen merchants, making them inefficient and unscalable. In contrast, the advanced language models, particularly pretrained large language models (LLMs) such as GPT[8], Llama[14], T5[10], and BERT[5], has demonstrated remarkable capabilities in analyzing complex data, offering a promising solution to the challenges. However, the application of LLMs in production environments comes with limitations. One significant drawback is the substantial computational resources required to run these models, which can lead to high operational costs and latency issues. Additionally, pretrained LLMs are not optimized for the unique requirements and nuances of particular applications, often necessitating extensive fine-tuning and optimization to perform well on specific tasks, which can be time-consuming and resource-intensive.

On the other hand, building a Transformer model from scratch can be necessary due to the unique and specialized nature for financial transaction data. Unlike general language data, transaction data often involves specific terminologies, unstructured formats, and domain-specific patterns that pretrained LLMs may not adequately capture. By developing a Transformer model from scratch, it is possible to design and train the model specifically for the intricacies of transaction data. This approach allows for the incorporation of domain-specific knowledge and the customization of the model architecture to better handle transactions.

Choosing between pretrained LLMs and small proprietary Transformer models can be a complex and challenging task in real-world applications. This decision involves weighing various factors such as accuracy, scalability, cost, and the specific requirements of the application at hand. Our study makes a significant contribution by demonstrating that while LLMs excel in general tasks, a proprietary



TRANSACTION DETAILS

SQ PANERA BREAD RYGU5IZY6R4

$56.30

Type	Sale
Transaction date	Aug 09, 2023
Posted date	Aug 10, 2023

Transaction Details

Panera Bread

$56.30

112.60 total rewards points earned

Transaction details

Type	Sale
Transaction date	Aug 09, 2023
Posted date	Aug 10, 2023

18423 Bryan Dr,
Rehoboth Beach, DE 19971

(302) 644-4414

Description	PANERA BREAD
Merchant type	Fast food
Method	In person

Category
Food & drink

Figure 1: Example for Transaction Understanding

model is essential for specific tasks, such as financial transaction understanding. We provide a comprehensive comparison of pretrained LLMs and small proprietary Transformer models, evaluating both accuracy and efficiency. Our findings reveal that larger models do not always outperform smaller ones in practical settings. Additionally, we offer detailed guidance on choosing the right proprietary models for real-world applications, assisting decision-makers in navigating the complexities of model selection.

2 Problem Definition

The aim of this study is to standardize POS (point-of-sale) transactions with merchant information, which provide value-added features for Chase customers and offer merchant-level insights for the business. Our goal is to identify the corresponding merchant IDs from messy transaction text and enhance the merchant information available to users on our Chase mobile app (as shown in Figure 1). Instead of displaying a raw transaction, we can present detailed merchant information, such as the logo, formal name, address, phone number, etc., to provide users with more context for inquiries and fraud detection.

The challenges stem from two main factors: the high volume and the highly disorganized nature of transactions. We need to process over 50 million transactions daily, requiring our model to respond within milliseconds while maintaining cost efficiency. Additionally, transactions are extremely complex and contain a significant amount of noise, making them difficult to manage with a simple solution. As illustrated in Table 1, numerous transactions lack an obvious sub-strings to identify merchants. For instance, "SWA * EARLYBRD XQQJWQ9V4F4" is difficult to associate with "Southwest Air". Furthermore, transactions often include various types of noise, such as aggregators (third-party online payment processors) like "SQ (Square)", but there is no consistent pattern for where and how these aggregators appear. Additionally, the presence of transaction numbers makes it more challenging. For instance, in "FAM EXP LDGX," the segment "LDGX" is a transaction number rather than part of the merchant's name.

[0]All transaction data mentioned in this paper is synthetic. Actual data is confidential

Our previous solution comprised two components: a set of regex rules (Rulebased) and an Enhanced String Distance (ESD) method. Rulebased method maintains thousands of regex rules to map incoming transactions to their corresponding merchant IDs. This method can only cover about 1000 merchants, which is highly unscalable and requires significant effort to create, modify, and maintain rules. The ESD method utilizes string similarity measures in conjunction with a manual decision tree, which is more flexible and can cover a larger number of merchants. However, the ESD method can only cover less than 20% of transactions, leaving a significant amount of transactions unmanaged. These transactions lead to poor user experiences and numerous transaction inquiry calls, which annually cost the company millions of dollars.

The purpose of this study is to explore advanced language models that can handle more complex transactions. Regarding advanced language models, Transformer-based models have emerged as the state-of-the-art (SOTA) in natural language processing (NLP). These models have demonstrated exceptional performance across a wide range of tasks due to their ability to capture intricate patterns and dependencies in data. In this study, we aim to explore both pretrained LLMs and small proprietary Transformer models. By comparing these approaches, we seek to determine the most effective solution for managing complex transactions, balancing performance, efficiency, and cost.

3 Methodology

LLMs, pretrained on extensive public data that includes merchant nicknames and abbreviations, have significant potential to standardize transactions. Alternatively, training a proprietary model from scratch with specific transaction data also promises accurate predictions. To investigate both models, we have established six work streams centered around three types of Transformer-based models: Encoder-Only, Decoder-Only, and Encoder-Decoder Models. We will compare the pretrained versions of these models with their proprietary counterparts.

3.1 Encoder Only Model

When utilizing the Encoder-Only model to address this problem, we employ the Bi-Encoder framework[11]. In this method, we initially use an encoder model to convert all merchants in our database into embedding vectors, which are then stored in a vector database. When a transaction occurs, the same encoder model is used to encode the transaction. We then search for and rank the most likely merchants based on similarity, selecting the top-ranked merchant as the match with Lucene Vector Search[19].

3.1.1 SentenceBert. The most widely used pretrained Encoder-Only model is SentenceBERT (SBERT) [11]. SBERT modifies the pretrained BERT network using Siamese and Triplet network[12], enabling faster and more accurate semantic search by deriving semantically meaningful sentence embeddings that can be compared using cosine similarity. We utilized the all-MiniLM-L6-v2[16] model from the MiniLM (Miniature Language Model) family. This model encodes both merchants and transactions into 384-dimensional dense vectors. During fine-tuning stage, we employed online contrastive loss to enhance the model's performance.

Transaction. Text	Transaction Zipcode	Merchant ID	Merchant Name
SQ * HM SP NTW P2FJOC4	12345	JPMC001	Home Shopping Network
AUTOMA MSFT * CORPO008	67890	JPMC002	Microsoft
SWA * EARLYBRD XQQJWQ9V4F4	13579	JPMC003	Southwest Air
FAM EXP LDGX	24680	JPMC004	Family Express
GOOGL * ADSJL0CZ1DN	98765	JPMC005	Google Ads

Data in the table is synthetic.

Table 1: Financial Transaction Examples

3.1.2 Proprietary Encoder-Only Model. We developed a proprietary encoder-only model using PyTorch by implementing the TransformerEncoderLayer module. This custom model is designed to encode both transactions and merchants with a unified encoder architecture. To train the model, we prepared a comprehensive dataset (see Section 4) and ensured it was properly tokenized (see Section 5.2.1). We configured hyperparameters, such as learning rate and weight decay, with Ray Tune [7] and Optuna [1], but for other model structure hyperparamters, we setup a series experiments to find the best ones(see Section 5). The model was trained using a contrastive loss function, as specified in Equation 1.

$$L = (1 - \text{pos\_sim}) + \max(0, \text{neg\_sim} - \text{margin}) \quad (1)$$

where the *pos_sim* indicates the cosine similarity between transaction and the positive merchant name, and *neg_sim* indicates the cosine similarity between transaction and the negative merchant name. We set margin as 0.5 on our data to achieve optimal results.

3.2 Decoder Only Model

When using the Decoder-Only model, we treat this transaction understanding problem as a translation problem, which generates a standardized merchant name from the messy transaction text. With the generated merchant name, we search our merchant database to find the most possible merchant ID with Lucene String Search.

3.2.1 Llama3-8b. Transaction data is confidential, so we only utilize open-source models that can be deployed internally, which is why we opted for Llama3 instead of GPT-4. Llama3[14] is the latest foundational LLM from Meta and achieves comparable results to GPT-4 in various benchmarks. Since the full version of Llama3 is unnecessary, we implement the Llama3-8b, which is optimized for dialogue use cases. Here is the given prompt to Llama3-8b:

> You are provided with raw merchant transaction text and the transaction's zipcode. Based on the provided information, please output the merchant name. Your answer should only include the predicted merchant name and nothing else.

followed with a user message with a transaction text and zipcode.

During the fine-tuning stage, we also include the corresponding merchant name in the assistant message. We applied QLoRA[4] to fine-tune the model.

3.2.2 Proprietary Decoder-Only Model. The proprietary Decoder-Only model is built using PyTorch's TransformerDecoderLayer module. In our implementation, the model takes transactions text as input and decodes them to desired merchant names as output. The model is trained in the similar way as described in Proprietary Encoder-Only Model part(Section 3.1.2). We applied Cross Entropy (Equation2) as the loss function to train the model.

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \quad (2)$$

where N is the number of tokens in the sequence, C is the vocabulary size.

3.3 Encoder-Decoder Model

The Encoder-Decoder model also translates transactions to merchant names, the same as Decoder-Only model does. The primary distinction between the Decoder-Only model and the Encoder-Decoder model lies in the additional encoding step present in the latter. Specifically, the Encoder-Decoder model employs several layers of encoders to transform a transaction into a context representation. The decoder then uses this representation, instead of transaction tokens, to generate output.

3.3.1 Flan-T5. Flan-T5[3] is a widely used pretrained Encoder-Decoder model, which is a family of models that enhances T5[10]. While fine-tuning, we follow the same settings to Llama3-8b.

3.3.2 Proprietary Encoder-Decoder Model. Similarly, we apply PyTorch's TransfromerLayer to build up our proprietary Encoder-Decoder model. The transaction data will first go through several layers of TransformerEncoderLayer and then leverage several layers of TransformerDecoderLayer to decode out the merchant name. The model training configuration and loss function is the same with Decoder-Only model (Section3.2.2).

4 Experiments

4.1 Transaction Dataset

We categorize our transaction data into three types: Rulebased Data, which consists of transactions managed by existing regex rules; ESD Data, which includes transactions handled by Enhanced String Distance (ESD) method; and Rawcleansed Data, which refers to transactions that are not managed by any existing methods and stored in their original form as they are received. Rulebased data takes about 63% of our dataset, ESD takes about 17%, and Rawcleansed takes about 20%.

- Rulebased Data: We conducted a query of our historical transactions and extracted a sample of approximately 1,131 rules from our dataset. For the training dataset, we compiled 773,653 transactions representing 779 merchants. Additionally, we prepared a testing dataset consisting of 1,311 transactions, also covering these 779 merchants.

- ESD Data: We queried our historical transactions and sampled 574,871 transactions managed by the ESD method, encompassing 506,135 merchants. For the testing phase, we prepared two distinct ESD test datasets. The first dataset, ESD_RD, includes 40,223 transactions involving merchants present in the training data. The second dataset, ESD_ZS, comprises 10,000 transactions involving merchants not previously seen in the training data.
- Rawcleansed: In the absence of ground truth for the Rawcleansed data, we manually labeled the merchants for a set of 2,541 transactions. This labeled dataset now serves as the ground truth for evaluating the model's performance on Rawcleansed data.

To train the Encoder-Only model effectively, the inclusion of negative samples is essential (shown as in Equation1). For each transaction, the corresponding matched merchant serves as the positive sample. We utilize Jaccard Similarity to compare the positive merchant with all other merchants. A merchant with a similarity score greater than 0.75 but less than 1.0 is randomly chosen as the negative sample. This approach ensures that the encoder model can differentiate between merchants that are similar but not identical.

4.2 Merchant Dataset

We compiled a dataset of 7.8 million merchants for matching purposes. To refine the candidate pool, we use the merchant's and transaction's zipcode as a filter. Following this, we apply either the generated merchant names or the encoded transaction vectors to the filtered candidates for more precise matching via Lucene, and select the top 1 merchant as the predicted match.

4.3 Evaluation Metrics

In our task, we use 'Accuracy' to evaluate models' performance. The Accuracy means the ratio of transactions that can be correctly map to a right merchant. Since we have four datasets, we apply a 'Weighted Accuracy' to evaluate each model's performance. The Weighted Accuracy is calculated as

$$\begin{aligned}\text{Weighted Accuracy} =& 0.63 * \text{Rulebased Accuracy} \\ &+ 0.085 * \text{ESD\_RD Accuracy} \\ &+ 0.085 * \text{ESD\_ZS Accuracy} \\ &+ 0.2 * \text{Rawcleansed Accuracy}\end{aligned}$$

where the weight for each part is derived from the percentage of this data present in our dataset.

5 Results and Discussions

5.1 Open Sourced or Proprietary Model

In this section, we evaluated three model streams: the out-of-the-box (OOTB) LLM, the fine-tuned LLM, and the proprietary model. To cover a broad range of LLMs, we selected Llama3 (decoder-only model), Flan-T5 (encoder-decoder model), and SBert (encoder-only model) as our LLM candidates. Correspondingly, we developed three types of proprietary models: a Proprietary Decoder-Only model, a Proprietary Encoder-Decoder model, and a Proprietary Encoder-Only model for comparison (shown as Figure2). The LLMs were fine-tuned with the exact same datasets used to train the proprietary models.

As shown in Figure 2, OOTB models are not well-suited for our internal uses cases. Although these models are designed for general purposes, they do not perform effectively on specific business scenarios because they lack exposure to the target data. After fine-tuning, their performance improves significantly since fine-tuning allows the models to adapt specifically to the nuances and patterns of the target data. Llama3's performance increases 9% and Flan-T5's performance increases 13% after fine-tuning. However, SBert's performance decreases 4%. The possible reason is that the transaction data is not well aligned with the model structure and original dataset so the transaction data introduces noises and biases that the model was not previously exposed to.

However, upon examining the proprietary models, we find that they can actually perform comparably or even better than LLMs with significantly fewer parameters. For instance, the Proprietary Decoder-Only model achieves 72.07% accuracy with just 1.7M parameters, compared to Llama3's 72.89% accuracy with 8B parameters. Proprietary Encoder-Decoder model performs better than Flan-T5 by 2% (70.14% vs. 68.31%) with much less parameters (1.5M vs 220M), and proprietary Encoder-Only model also performances better than SBert by 6% (60.06% vs. 54.63%) with smaller size (11M vs 22.7M). This suggests that proprietary models can be more efficient in certain tasks compared to larger pretrained models. Table 2 [1] provides detailed insights into the speed and cost of training/fine-tuning each of these models. It is evident that proprietary models are smaller and faster in both training and inference, resulting in budget savings. Given that we need to process over 50+ million transactions per day, a 100-millisecond upper limit for inference time is crucial. Although Llama3 achieves 0.8% higher accuracy, it requires more than 700 milliseconds to infer one transaction. In contrast, the Proprietary Decoder-Only model takes only 95.02 milliseconds to achieve comparable accuracy, offering much faster inference time and reduced costs. Proprietary Decoder-Only model achieves better accuracy compared to the Proprietary Encoder-Decoder model (72.07% vs. 70.14%), but it is significantly slower in inference despite having similar model sizes. The possible reason is that the encoder-decoder model architecture can efficiently utilize parallel processing during the encoding phase and make use of the encoded context for effective decoding. In contrast, the decoder-only model architecture has to access a rich context from the start, which increases the complexity of each decoding step. Additionally, in a decoder-only model, the entire process of understanding the input and generating the output is handled in a single pass. This requires the model to rely on previously generated tokens to predict the next one, known as autoregressive generation, which can slow down the overall process. Encoder models are faster and cheaper, even with much larger model size. This is because the encoding process can be well paralleled. However, their performance on this task is not as good as models with decoders.

[1]The price is estimated according to the AWS pricing page: https://aws.amazon.com/sagemaker-ai/pricing/

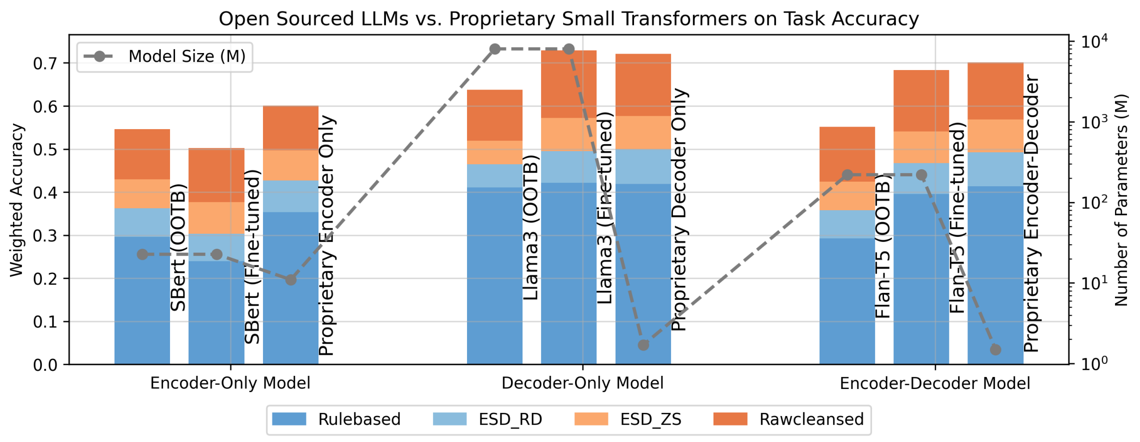


Figure 2: Open Sourced LLMs vs. Proprietary Small Transformers on Task Accuracy

Model	Model Size	Machine	Training Speed	Training Cost	Inference Time	Inference Cost
Llama 3	8B	Training : g4dn.12xlarge Inference: g4dn.2xlarge	166h/1M txn	$ 812/1M txn	735 ms/txn 204.17h/1M txn	$191/1M txn
Flan-T5	220M	Training : g4dn.12xlarge Inference: g4dn.2xlarge	108h/1M txn	$528/1M txn	284.7 ms/txn 3.52h/1M txn	$22.56/1M txn
SBert	22.7M	g4dn.xlarge	1.7h/1M txn	$1.60/1M txn	5.77 ms/txn 1.60 h/1M txn	$1.51/1M txn
Proprietary Decoder Only	1.7M	g4dn.xlarge	0.33h/1M txn	$0.24/1M txn	95.02 ms/txn 26.39h/1M txn	$19.43/1M txn
Proprietary Encoder-Decoder	1.5 M	g4dn.xlarge	0.11h/1M txn	$0.08/1M txn	26.50 ms/txn 7.36h/1M txn	$5.42/1M txn
Proprietary Encoder Only	11.3M	g4dn.xlarge	0.46h/1M txn	$0.34/1M txn	7.62 ms/txn 2.11h/1M txn	$1.57/1M txn

Table 2: Open Sourced LLMs vs. Proprietary Small Transformers on Speed and Cost

5.2 The Larger the Better?

From the results in Figure 2, we are prompted to ask whether a larger model necessarily leads to better outcomes. We use a series of experiments to explore this question. Speaking of model size, three factors contribute to it, the vocabulary size, and number of layers, and embedding size. So, we discuss from these three dimensions.

5.2.1 Tokenizer and Vocabulary Size. A tokenizer is a key part of Transformer-based models, essential for preparing text data. It converts raw text into a format the model can process by breaking it down into smaller units called tokens, which can be words, subwords, or characters. Common tokenizers include BPE, WordPiece, and Unigram. BPE[13] starts with individual characters, and merge the most frequent pairs of characters, until pre-defined vocabulary size is reached. WordPiece[18] starts with individual characters and adds the most frequent subword units to the vocabulary to maximize the training data's likelihood. Unigram[6] uses a probabilistic model treating subword units as independent tokens. It starts with a large vocabulary and removes the least likely units to maximize the training data's likelihood. Given that there is no definitive answer regarding the best tokenizer or the optimal vocabulary size, we address this issue by experimenting with all three popular tokenizers mentioned above, and vary the vocabulary size from 100 to 10,000 to check whether a larger vocabulary yields better results.

Figure 3 addresses both questions, emphasizing the importance of selecting the right tokenizer and vocabulary size for optimal performance. The Encoder-Only model performs best with WordPiece and significantly worse with Unigram. On the other hand, the Encoder-Decoder model yields better results with BPE compared to WordPiece. The Decoder-Only model seems less affected by the choice of tokenizer, performing well with all three options.

The Encoder-Only model benefits from WordPiece because WordPiece is designed to create a balance between vocabulary size and token granularity. It segments words into subword units that are frequent enough to capture meaningful linguistic information. In contrast, Unigram uses a probabilistic approach to select subword units, which sometimes leads to less optimal segmentation and produce a larger variety of subword units that are not meaningful. BPE is designed to iteratively merge the most frequent pairs of bytes, resulting in a compact set of subword units. This compression is beneficial for the Encoder-Decoder models, which need to handle both input and output sequences efficiently, especially when transforming sequences of similar lengths. The Decoder-Only model's insensitivity to tokenizer choice suggests its architecture is robust enough to handle various tokenization strategies without significant performance changes.

Regarding vocabulary size, bigger is not always better. The Encoder-Only model achieves its best performance with a vocabulary size of 1000, which might provide the right balance between having enough subword units to capture nuances and not overwhelming the model with too many options, which can complicate learning. While both the Decoder-Only and Encoder-Decoder models perform optimally with a smaller vocabulary size of 500, because it reduces the complexity of the output space, making it easier for the model to learn and generate sequences effectively.

In our subsequent experiments, we use the WordPiece tokenizer with a vocabulary size of 1000 for the Encoder-Only model and BPE tokenizers with a vocabulary size of 500 for both the Encoder-Decoder model and Decoder-Only model.

5.2.2 *Model Size.* After setting up the vocabulary and tokenizer, we will discuss about whether a model with more parameters will yield better results. We first vary the embedding dimension for the three models from 16 to 1024 while keeping the model depth fixed at 8. As shown in Figure 4(a), increasing the embedding size generally enhances model performance. However, the Encoder-Only model is particularly sensitive to this parameter, as it requires a wide embedding size to effectively store context information. The Encoder-Only model achieves its best performance at an embedding size of 512, after which performance declines with larger sizes. Similarly, the Decoder-Only model's performance improves initially but drops significantly when the embedding size exceeds 256. The Encoder-Decoder model, on the other hand, demonstrates stability across different embedding sizes, although its accuracy slightly increases and then begins to decline after reaching 128.

We vary the number of layers from 2 to 16 for the three models. In the Encoder-Decoder model, layers are evenly split, so an 8-layer Encoder-Decoder model has 4 encoder layers and 4 decoder layers. As shown in Figure 4(b), the Encoder-Only model is particularly sensitive to this parameter, with accuracy increasing dramatically at first, peaking at 8 layers, and then declining. In contrast, the Decoder-Only and Encoder-Decoder models exhibit greater stability with respect to the number of layers, indicating that model depth has less impact on their performance.

Figure 4(c) provides a comprehensive overview of the models' performances relative to their sizes, which are determined by either the model's width (embedding size) or depth (number of layers). The results indicate that the Encoder-Only model benefits from a larger model size, achieving better performance with increased width or depth. In contrast, the Decoder-Only model performs better with small-to-medium model sizes, as larger sizes tend to introduce noise rather than useful information. The Encoder-Decoder model demonstrates greater stability across different model sizes.

5.2.3 *Model Type.* We have tried three different types of Transformer model in our task, namely Encoder-Only, Decoder-Only, and Encoder-Decoder model. As shown in Figure3 and Figure4, Decoder-Only and Encoder-Decoder models outperform Encoder-Only models. Decoder-Only model and Encoder-Decoder model benefit from their autoregressive nature in the decoder part, which allows them to generate tokens sequentially and maintain coherence, making them particularly effective for tasks like text generation. When comparing the Encoder-Decoder model to the Decoder-Only model, the Decoder-Only model slightly outperforms the Encoder-Decoder model. This performance difference may be attributed to the potential information loss that occurs during the encoding process, leading to less accurate decoding compared to the Decoder-Only model, which does not undergo this encoding step.

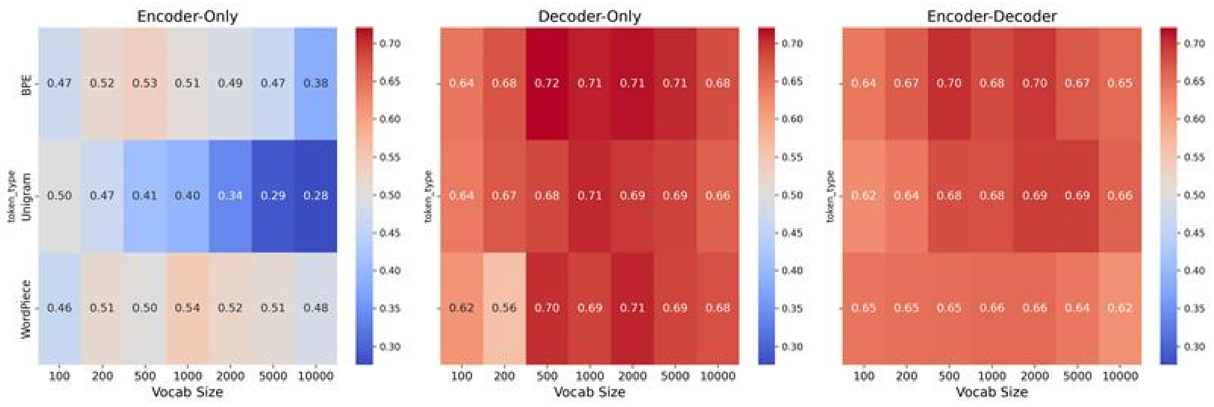
Table 3 presents a comparison among the three types of proprietary models. For a fair evaluation, we selected the best-performing model from each category. The Encoder-Only model converges quickly, requiring only 899 iterations, while the Decoder-Only model takes longer to train, converging after 10,499 iterations. Despite the Encoder-Only model being larger, both models take a similar amount of time to converge. During inference, the Encoder-Only model is significantly faster, even with its larger number of parameters. However, the Encoder-Only model requires a longer time for the vector level indexing, and the index must be rebuilt each time the model is updated. In contrast, the Decoder-Only and Encoder-Decoder model are quicker to index and does not require re-indexing upon model updates, as the indexing is performed at the merchant level. Although both models are slower during inference, they still satisfy our 100ms/txn requirement.

If we investigate the Accuracy on each type of dataset, all models perform comparably better on ESD data compared to Rule-based data and Rawcleansed data. This indicates that the model effectively leverages similar patterns within the data for matching purposes. The performance on Rulebased transactions falls significantly,which is likely due to that Rulebased data contains business logic that could be captured by string-level patterns. The Rawcleansed data is unseen in our training data and of higher complexity, but Decoder-Only model can still get 72% accuracy, dramatically higher than the other two models. This highlights the potential to utilize the Decoder-Only model to improve the coverage of our transaction system.

6 Model Deployment

Since the Decoder-Only model achieves 72% accuracy on Rawcleansed data, and our business priority is expanding coverage on Rawcleansed data. In the initial deployment, we use the Proprietary Decoder-Only model for Rawcleansed data data while retaining the Rulebased and ESD components for further enhancement.

The deployment pipeline is illustrated in Figure 5. For POS transactions, we first apply general string cleaning, including lowercase conversion and non-alphanumeric removal. The transaction then passes through our Rulebased matching and ESD matching components. Transactions that do not find a match, known as Rawcleansed transactions, are processed by the Proprietary Decoder-Only model via API calls. The model outputs a confidence score and a generated merchant name for each transaction. We then use the generated merchant name for a Lucene search to identify the top merchant. However, this top merchant is only correct 72% of the time, and we aim to avoid showing incorrect predictions to users. Therefore, we use the similarity between the top merchant's name and the generated merchant name, along with the model's confidence score,



The number showing in the heatmap is the Weighted Accuracy with the corresponding tokenizer and vocabulary size. To make a fair comparison, all the models are using similar configurations with 8 layers and 128 embedding size.

Figure 3: Models' performance on with different tokenizers and vocabulary size

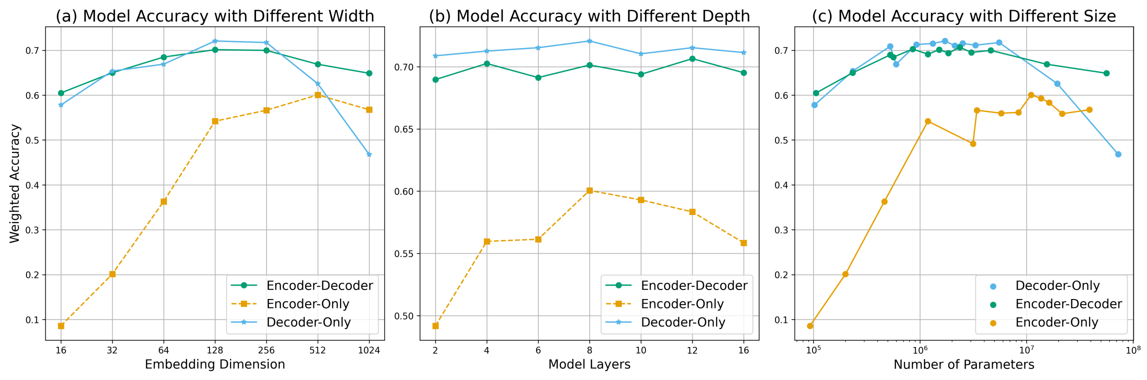


Figure 4: Models' performances with different Model Size

Model Type	Tokenizer	V	D	L	N	Train Iters	Train Time	Index Time	Infer Time (/txn)	Accuracy				
										Top-1	Top-5	Precision	Recall	F1
										Rule	ERD	EZS	Raw	WT
Decoder Only	BPE	500	128	8	1.72M	10499	2.3 hrs	6.5 mins	95.02 ms	0.66	0.95	0.91	0.72	0.72
Encoder Decoder	BPE	500	128	8	1.52M	4599	1.3 hrs	6.5 mins	26.50 ms	0.66	0.92	0.90	0.66	0.70
Encoder Only	WordPiece	1000	512	8	11.03M	899	2.2 hrs	80 hrs	11.62 ms	0.56	0.87	0.82	0.52	0.60

V indicates the vocabulary size, D indicates the embedding dimension, L indicates the number of layers, and N indicates the number of parameters. Rule is the Accuracy on Rulebased dataset, ERD is the Accuracy on ESD_RD dataset, EZS is the Accuracy on ESD_ZS dataset, and Raw is the Accuracy on Rawcleansed dataset, and WT is the final Weighted Accuracy. All these models are trained with the same batch size and similar learning rate.

Table 3: Comparison Among Proprietary Models

as filters to eliminate incorrect predictions. Monthly, we manually sample and review results, gathering incorrectly matched and inquired transactions to fine-tune the model.

Table 4 provides an overview of the business metrics following the deployment of the Decoder-Only model. Currently, we retained the Rulebased and ESD components while focusing solely on the Rawcleansed data, which our previous system could not process. Rawcleansed data accounts for 20% of our transactions, and the

Your First Column	Transaction Coverage				Cost Reduction
	Rule	ESD	Raw	All	
Before	63%	17%	0%	80%	–
After	63%	17%	14%	94%	$13.2M/year

Table 4: Business Metrics After Decoder Model Deployment

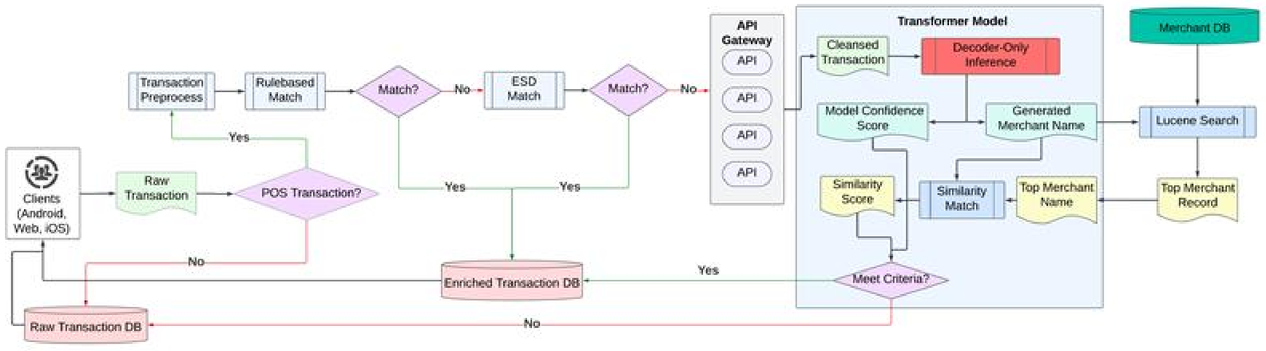


Figure 5: Model Deployment

Decoder-Only model accurately predicts 72% of these transactions. This leads to a 14% increase in processed transactions, raising our overall transaction coverage from 80% to 94%. As a result, the improved coverage helps reduce transaction inquiry calls, generating an estimated annual savings of approximately $13.2 million.

7 Related Work

Transformer models have revolutionized the field of natural language processing (NLP) since their introduction[15]. These models use self-attention to capture sequence dependencies, enabling efficient and accurate language processing. They fall into three types: Encoder-Only, Decoder-Only, and Encoder-Decoder models..**Encoder-Only Models** such as BERT[5], focus on understanding the input sequence by capturing contextual information from both directions. A notable example of an encoder-only model is SBERT[11], which modifies BERT to generate semantically meaningful sentence representations. **Decoder-Only Models** such as GPT[2, 9], are designed for text generation tasks. These models generate text by predicting the next token in a sequence, leveraging autoregressive mechanisms. Decoder-only models excel in tasks like text completion and dialogue generation.**Encoder-Decoder Model** such as T5 (Text-To-Text Transfer Transformer), combine the strengths of both encoders and decoders. The encoder processes the input sequence to generate a context representation, which the decoder then uses to generate the output sequence. This architecture is particularly effective for tasks like machine translation, summarization, and question answering.

Large Language Models (LLMs) are advanced AI systems with vast parameters and training data, enabling them to capture complex linguistic patterns. Built on Transformer architecture, they excel in diverse NLP tasks across various fields. LLM integration into finance also becomes a significant area of research and exploration. Yang et al. introduced FinBERT[21], a BERT-based model fine-tuned for financial analysis, demonstrating its effectiveness in capturing the nuances of finance market. Bloomberg released BloombergGPT[17], a LLM specifically designed to enhance financial NLP tasks. Additionally, Xiao-Yang Liu et al. presented FinGPT[20], aimed at enhancing tasks such as market prediction and financial data interpretation. However, much of this research has focused on developing general financial LLMs or applying them to general tasks like sentiment analysis or named entity resolution. Few studies have concentrated on specific tasks such as Financial Transaction Understanding. The Slope company[2] has proposed two LLM based transaction understanding models: SlopeGPT[3] and SlopeTransFormer[4], but both models are fine-tuned from existing LLMs, and they do not provide detailed accuracy comparisons or explore the potential of using smaller models to achieve better performance.

8 Conclusion

This paper presents a comprehensive comparison between LLMs and small proprietary models within the domain of financial transaction understanding. We conducted extensive experiments on three types of Transformer models: Encoder-Only, Decoder-Only, and Encoder-Decoder, using both LLMs and small proprietary models. Our results indicate that small proprietary models offer comparable accuracy, faster processing and lower costs, making them more suitable for real-time financial applications. These findings emphasize the importance of selecting models based on domain-specific needs and highlight the potential benefits of customized proprietary models in specialized settings. In production, we chose to deploy the Proprietary Decoder-Only model to handle previously unmanageable complex transactions. This improved transaction coverage by 14% and reduced annual costs by over $13 million.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research* 25, 70 (2024), 1–53.

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems* 36 (2024).

[2]https://slopepay.com/company
[3]https://medium.com/slope-stories/slopegpt-the-first-payments-risk-model-powered-by-gpt-4-cd444ab5242d
[4]https://medium.com/slope-stories/slope-transformer-the-first-llm-trained-to-understand-the-language-of-banks-88adbb6c8da9

[5] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, Vol. 1. Minneapolis, Minnesota, 2.

[6] Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959* (2018).

[7] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118* (2018).

[8] Alec Radford. 2018. Improving language understanding by generative pre-training. (2018).

[9] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[11] N Reimers. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).

[12] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[13] Rico Sennrich. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909* (2015).

[14] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[15] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[16] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems* 33 (2020), 5776–5788.

[17] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564* (2023).

[18] Yonghui Wu. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[19] Jasper Xian, Tommaso Teofili, Ronak Pradeep, and Jimmy Lin. 2024. Vector search with OpenAI embeddings: Lucene is all you need. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 1090–1093.

[20] Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. 2023. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031* (2023).

[21] Yi Yang, Mark Christopher Siy Uy, and Allen Huang. 2020. Finbert: A pretrained language model for financial communications. *arXiv preprint arXiv:2006.08097* (2020).