Developing a Large Tabular Model for Predicting Credit Approvals and Analyzing Customer Behavior in Financial Institutions

Final Report

Atharva Kulkani, Minh Dang, Nicolás Corgono, Ryu Sonoda, Varun Agarwal, Yu Zheng Lim

Dec 19, 2024

Executive Summary

This project explores the use of Tabular Foundational Models (TabFM), specifically Llama2-GTL (Generative Tabular Learning), for predicting credit approval. While traditional machine models have demonstrated effectiveness in predicting customer behavior, they often require significant retraining and feature engineering, which can be resource-intensive and time-consuming. GTL models, leveraging zero-shot and in-context learning, offers the potential for a more efficient approach by reducing the need for extensive retraining and manual preprocessing.

The project first establishes baseline models using traditional machine learning algorithms, including logistic regression, decision trees, random forest, and XGBoost. Various preprocessing techniques, such as feature engineering and SMOTE for handling data imbalance, are employed.

In the next phase, the performance of Llama2-GTL is evaluated using pre-trained versions without fine-tuning. Different models, including the 7B (quantized and unquantized) and 13B (quantized) versions were tested using different prompt types (T-table vs. T-anony). The results show that the T-table prompt technique, which provides more information to the model, consistently outperforms the T-anony prompt. Additionally, the larger 13B model outperforms the smaller 7B model, underscoring the importance of model size in achieving better performance. The project further reveals that balanced in-context examples (equal number of majority and minority class examples) when prompting improves F1 scores. A larger sample size with fewer features also yields better results compared to a smaller sample size with more features, adhering to the context length constraints of large language models like Llama2.

Finally, comparing Llama2-GTL against the traditional models, the 13B-GTL-8bit with the T-table prompt type performed on average slightly worse than logistic regression but outperformed decision trees, random forests, and XGBoost. However, the stable performance across varying sample sizes positions Llama2-GTL as a promising tool for future credit approval prediction tasks, offering significant computational and efficiency advantages over traditional methods.

In conclusion, Llama2-GTL shows potential for improving financial predictions with reduced retraining, marking a step forward in utilizing generative learning in financial applications.

1. Problem Statement

Financial institutions, like our industry partner TD Bank, are increasingly harnessing data science to enhance customer service, manage risks, and optimize operational efficiency. A major challenge lies in understanding and predicting customer behaviors and characteristics. This project aims to develop a tabular foundational model (TabFM) or a Large Tabular Model (LTM) for bank customers by leveraging publicly available data[1], to predict if a customer will get approved for a credit product.

Currently, these can be done by traditional models like logistic regression and decision trees [1], which have been supplemented by advanced techniques such as gradient boosting [2] and deep learning [3, 4]. However, they often require extensive retraining for new tasks. On the other hand, TabFM leverages zero-shot and in-context learning, enabling it to adapt to new tasks with minimal retraining, thus improving model efficiency and effectiveness.

This research integrates advanced methodologies in generative learning with practical financial applications. Unlike traditional approaches that require preprocessing steps such as feature engineering, imputation, and retraining, Generative Tabular Learning (GTL) models harness their pretrained architectures to achieve competitive results on tabular data tasks. We examine their performance against baseline TradML models—Logistic Regression, Decision Trees, Random Forests, and XGBoost—under similar constraints of limited data and features to ensure a fair evaluation.

Our overall approach follows these stages:

1. Data preparation: Gather, explore, and preprocess the dataset. Split it into training and test sets.
2. Model Development: Build baseline models. Apply traditional ML models such as logistic regression and XGBoost.
3. GTL Implementation:
    a. Use zero-shot learning for initial predictions.
    b. Apply in-context learning with example prompts for task adaptation.
4. Evaluation: Compare GTL's performance with baseline models using sensible metrics.
5. Documentation and Knowledge Transfer

2. Literature Review

The main research paper that TD advised us to focus on, published by Wen et al [1] from Microsoft, has been the core reference for our work.

[1] https://www.kaggle.com/datasets/saurabhbadole/leading-indian-bank-and-cibil-real-world-dataset

Abstract

Generative Tabular Learning (GTL) brings the powerful capabilities of large language models (LLMs) to tabular data, tapping into features like zero-shot generalization and in-context learning. By continuing to pre-train on 384 diverse datasets, GTL enables LLMs to understand numerical relationships and domain-specific knowledge. The approach is built on Llama-2 and is fine-tuned for performance in tabular tasks.

Introduction

Ensemble and tree-based models have been the go-to for predictive tasks, but they fall short when it comes to creating models that can generalize across different datasets with limited labeled data. This is where universal models are needed — models that can easily transfer to new datasets with just a few labeled examples. Pre-training deep learning models specifically for tabular data is a promising approach to fill this gap.

Two standout features of these models are: (1) After pre-training, they can quickly adapt to new tasks by using task-specific prompts, which allows for efficient knowledge transfer, and (2) they have in-context learning capabilities, meaning they can learn from a few labeled examples without needing to be fine-tuned.

Most previous work has focused too heavily on fine-tuning, missing out on key benefits like in-context learning and few-shot generalization. On top of that, traditional models struggle to handle tabular data formatted as language and often fail to incorporate critical domain knowledge.

GTL Framework

GTL introduces a pipeline that converts raw tabular data into an instruction-based format, designed to capture the relationships between numerical features and targets while also including meta-information. This approach is key to enabling zero-shot and few-shot learning without needing to update the model's parameters. For example, Llama-GTL achieves state-of-the-art performance in several few-shot learning scenarios.

While existing models, like TabPFN, focus mainly on classification, GTL goes beyond this by introducing Language-Interfaced Fine-Tuning (LIFT), which extends its use for more complex tabular tasks through language-based templates.

GTL for LLMs

The GTL process for converting tabular data into an instruction-based format consists of nine steps. It starts with gathering datasets from domains like shopping, health, and finance, followed by organizing the data into tables that include necessary details like columns and value

explanations (optional). The problem at hand, whether classification or regression, will then be formatted for either zero-shot or in-context learning.

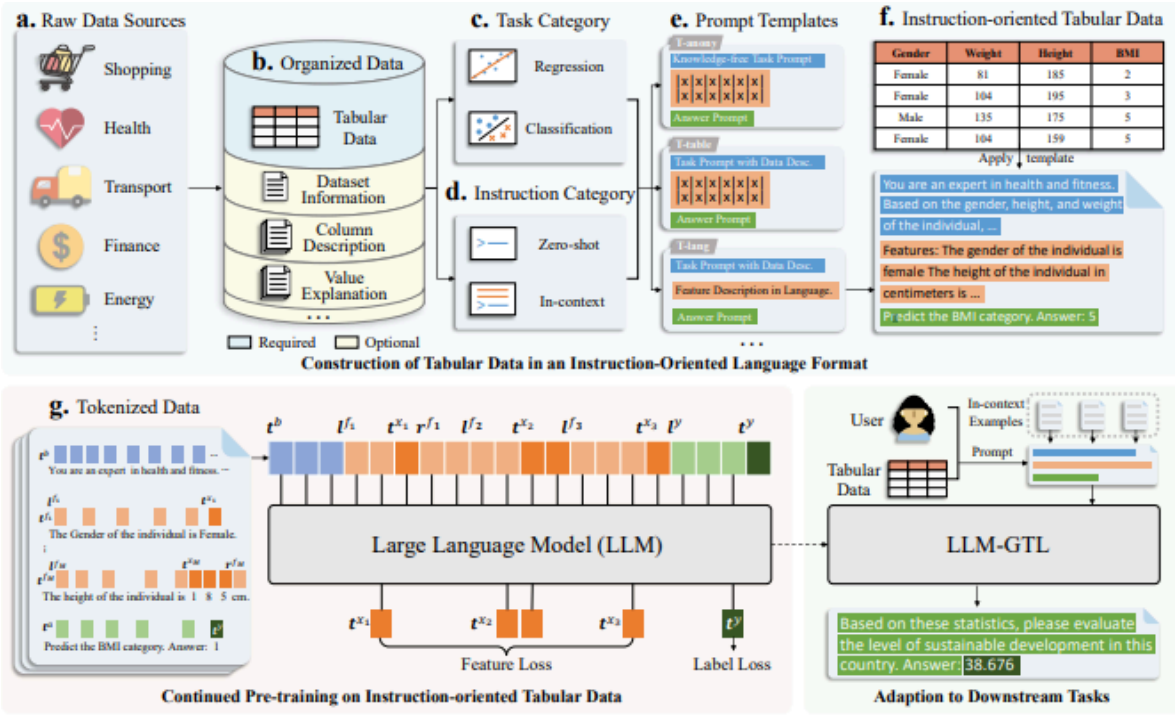

Figure 1: Overall approach: GTL Pipeline for tabular data construction

The pipeline is briefly described as follows:

1. A variety of tabular datasets from different domains is collected, covering a broad range of predictive tasks.
2. For each task, the data and its meta-information are transformed from tabular format into a language-based, instruction-oriented format. This format includes:
    a. Task instructions that explain the background and goal of the task.
    b. Feature descriptions that provide details on the feature values.
    c. Answer descriptions that specify the prediction targets.
3. Pre-train an LLM on that data, to produce a GTL-enhanced LLM, or a TabFM.

Templates for Data Formatting

To further simplify the process, GTL uses three distinct templates for formatting data:

T-table template: Displays data in a table using Markdown. This template is more efficient for handling in-context examples, although it may be more challenging for LLMs when it comes to long-term associations.
T-anony template: Similar to T-table, but it removes any meta-information, making it useful when background data isn't available.

These templates make GTL adaptable to a range of tasks and datasets, giving flexibility when it comes to using meta-information and including in-context examples.

Probabilistic Framework

GTL uses a probabilistic framework that captures relationships between features and target variables, taking into account the meta-information to give context. It works by representing the joint probability distribution of meta-information and text representations for features and targets, allowing GTL to model complex dependencies and nuances that traditional models might miss.

$$p(\boldsymbol{x}, y) = p(\boldsymbol{t}^{\boldsymbol{x}}, t^{y} | t^{m}) = p(t^{y} | \boldsymbol{t}^{\boldsymbol{x}}, t^{m}) \prod_{i=1}^{M} p(t^{x_i} | t^{<x_i}),$$

The first term represents joint probability distribution conditioned on meta information (t^m) and using text rep for features (t^x) and (t^y) for target.

For the second equation: p(t^y | t^x, t^m) represents the probability of the target text representation given the feature text representations and meta-information and the summation part represents the probability of each feature text rep given all previous feature reps. Through this, GTL captures complex dependencies between target and features, models relation between features themselves and incorporates meta- information to provide context.

By using this probabilistic framework, GTL can generate predictions for new tabular data samples and potentially learn intricate patterns in the data that traditional tabular models might miss.

Zero-Shot and In-Context Learning

One of the most exciting features of GTL is its ability to handle zero-shot learning. This means it can make predictions for tasks it hasn't seen before, even when the data schema is unfamiliar, as long as meta-information is provided. In-context learning takes this a step further by introducing a few examples from the new task, allowing the model to make better predictions. With minimal fine-tuning, this can transition into few-shot learning, making GTL highly adaptable and efficient across different tasks.

3. Exploratory Data Analysis and Preprocessing

The literature review lays the groundwork for the work in the second half of our project: using TabFM to predict whether a customer would be approved for credit products, and to answer other business questions by predicting other target variables. The sections below address the

first half of the project: to establish baseline results from traditional ML baseline models, for TabFM to be compared against.

Dataset

This project utilizes three datasets: Internal Banking, External Banking, and an Unseen Dataset sourced from Kaggle.

1. **Internal Banking Dataset**: This dataset contains proprietary information collected and maintained by a bank, encompassing a wide range of customer-related data. It includes customer's loan portfolios, credit card usage, and other operational and financial metrics relevant to the bank's internal activities. It contains 51,336 rows and 26 features.
2. **External Banking Dataset**: The external dataset complements the internal data by providing additional credit information sourced from the Credit Information Bureau (India) Limited (CIBIL). This dataset includes detailed credit profiles, such as credit scores, loan repayment behavior, outstanding debts, and historical credit data, offering a broader view of customers' financial activities beyond the bank's internal systems. It contains 51,336 rows and 62 features.
3. **Unseen Dataset**: The unseen dataset is a small subset (100 rows, 42 features) derived from both the internal and external data, meant for model testing. However, it is limited in scope, as it does not contain the full feature set present in the other two datasets, hence prediction models trained using features from Internal and External may not work for Unseen. Due to its incomplete nature and small size, this dataset will not be used in the project. We instead merged the first 2 datasets, then derived the training and testing sets from that.

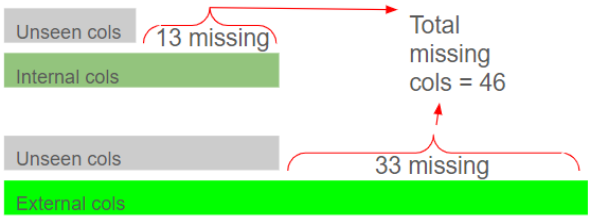

Figure 2: Missing columns in Unseen dataset

Target Variable

For the purpose of Exploratory Data Analysis (EDA) and Modeling, we have combined the internal and external datasets using *PROSPECTID* column (primary key) and we will be using the *Approved_flag* feature as the target variable for analysis. *Approved_flag* takes 4 values: P1, P2, P3, P4 with P1 signifying the least credit risk and P4 denoting the most risk and having the maximum chance of rejection.

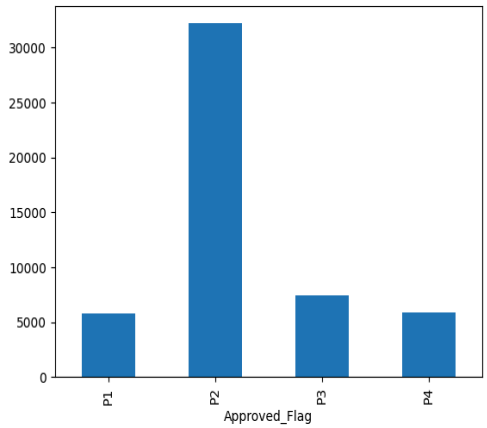

Figure 3: Countplot of Approved flag across different tiers

The plot highlights a significant imbalance in the target variable, with a noticeably higher count for the P2 flag compared to the other categories.

Per TD's requirements, we reframed the problem by converting it from a multiclass classification into a binary classification, where the output will indicate either 'Approved' or 'Not Approved.' Specifically, the P1, P2, and P3 flags will be grouped under 'Approved' (Class 0), while P4 will be deemed as 'Not Approved'[2] (Class 1). Since the project's objective is to accurately classify customers into these two broader categories, the individual flag distinctions will not be considered.

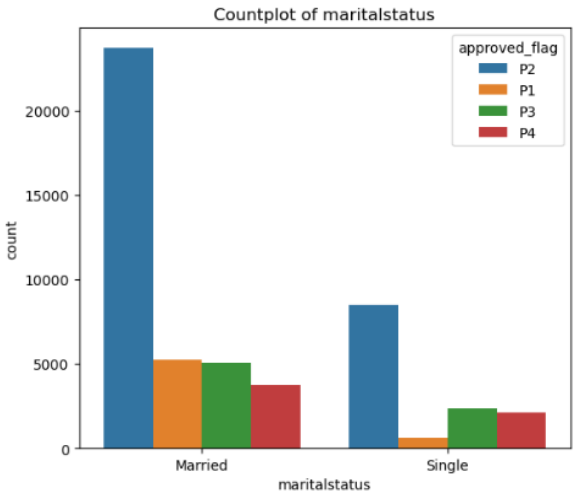

Fig. 4: Distribution of Approved Flag w.r.t Marital Status

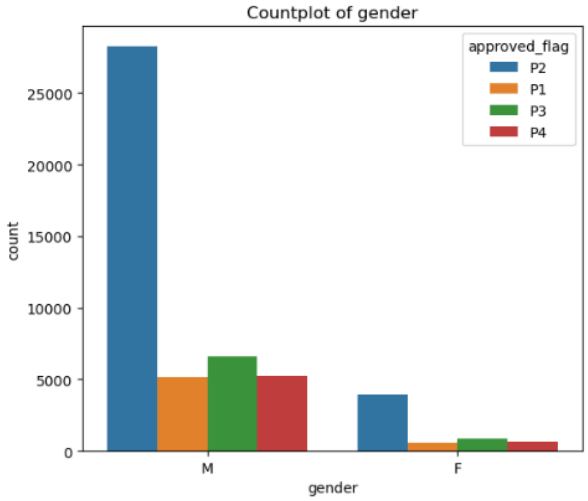

Fig. 5: Distribution of Approved Flag w.r.t Gender

The analysis reveals a significant imbalance in the dataset in terms of both gender and marital status, with a notably higher proportion of males compared to females, and a greater number of

[2] Due to the lack of online information on the definitions of P1-4, we made this decision through consulting ChatGPT (which holds global knowledge from past till minimally 2021) and our industry mentor from TD

married individuals compared to singles. However, despite these disparities, the distribution of the target variable within each gender category remains relatively similar.

The dataset reveals a notable prevalence of individuals with graduate degrees. In terms of age distribution, over 45% of participants fall within the 26 to 35 age range. Additionally, approximately 44% of the respondents have an annual income between $18,000 and $30,000.

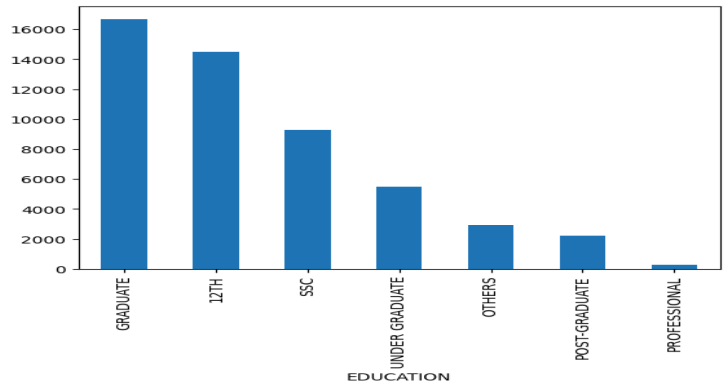

Fig. 6: Countplot for Education Variable

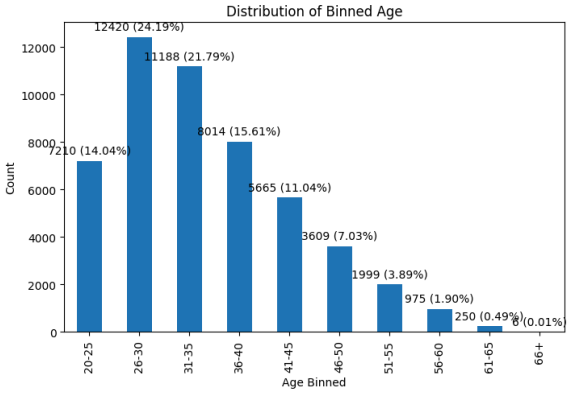

Fig. 7: Countplot for Age Variable

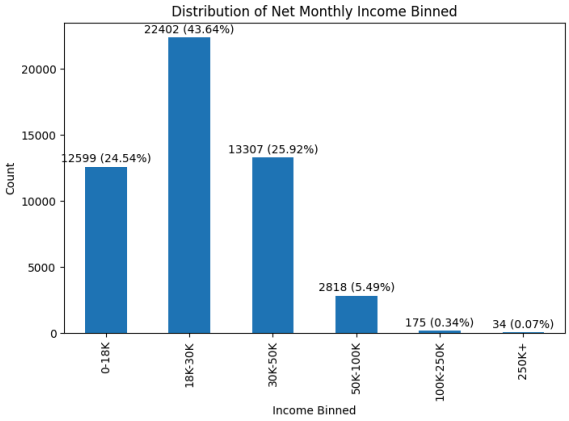

Fig. 8: Countplot for Income Variable

<u>-99999 and missing values</u>

During our analysis of the dataset, we found multiple columns containing a significant number of “-99999” values. Specifically, eight features exhibited over 20% of these values, with *cc_utilization* and *pl_utilization* showing the highest percentages at 92.8% and 86.6%, respectively.

Column	Error Percentage
cc_utilization	92.792582
pl_utilization	86.557192
time_since_recent_deliquency	70.026882
max_delinquency_level	70.026882
time_since_first_deliquency	70.026882
max_unsec_exposure_inpct	45.149603
max_deliq_6mts	25.109085
max_deliq_12mts	21.100203

Fig. 9: Features with high number of -99999 values

It is worth noting that “-99999” has an underlying meaning (eg. missing value indicator, and not necessarily 0), since there are already ‘0’ values present in many of these variables. We also noted that it is the only negative value found across all variables.

We distinguished two types of “-99999” columns: time-based (eg. *time_since_first_deliquency, time_since_recent_deliquency, time_since_recent_payment*) and quantity-based features (eg. *tot_enq, CC_enq, CC_enq_L6m*). We explored several possible combinations for preprocessing and concluded that the following process achieves the highest performance: For the time-based variables, such as time_since_first_deliquency, we created a new dummy column that takes value 0 when the variable has “-99999” value and 1 otherwise. For the quantity-based variables, such as "PL_enq" (number of personal loan queries), the zero value is already present on the column indicating that “-99999” necessarily has another meaning. For such values and features with many missing values, we used k-means to impute them.

<u>High percentage values exceeding 100%</u>

Additionally, we identified a few percentage-type columns containing observations with values exceeding 100%. For example, *pct_currentBal_all_TL* (which reflects the percentage of current balances across all accounts) and *max_unsec_exposure_inPct* (representing maximum unsecured exposure as a percentage). We kept them, as it is possible some of these people took huge loans.

4. Modeling

Due to the imbalance of the target variable, we used the F-1 score as the primary benchmark for all the models. Hyperparameter tuning for each model was conducted using 5-fold randomized cross-validation. The tuned hyperparameters for each model are summarized in Table 1.

Model	Tree based	Ensemble tree	Logistic Regression
Hyperparameter	Max depth Min sample split Min sample leaf Criteria for split	# of estimators Max depth Min sample split Min sample leaf	Regularization Weight Regularization type Solver Max iteration

Table 1: Model Hyperparameters

Initial Baseline

As an initial baseline model, we constructed a random forest by applying only one-hot encoding to all categorical variables (“minimal preprocessing”). This yielded an F-1 score of 0.696. After applying the preprocessing steps described earlier (“best preprocessing”), the F-1 score slightly improved to 0.706 (Table 3).

Feature importance analysis (Fig) revealed that the variables related to enquiries were the most influential, while the time since the last delinquency or the age of the oldest account were also relevant, aligning with our prior expectations.

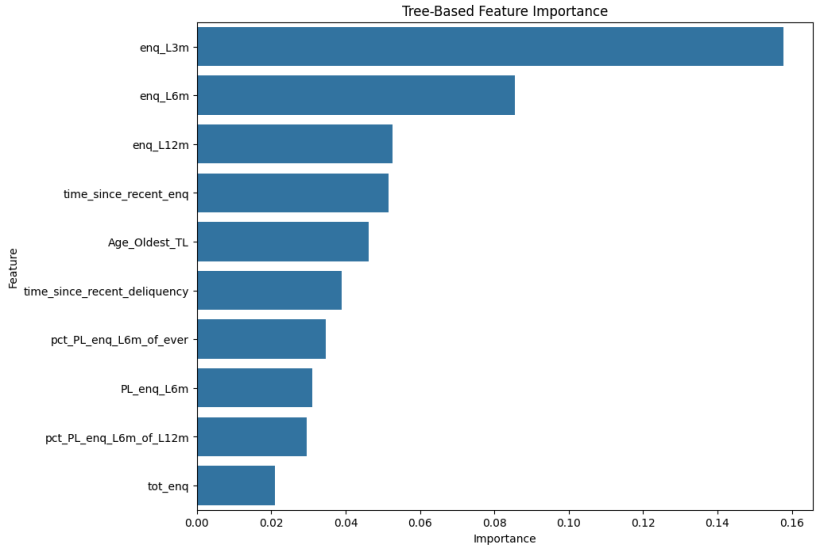

Fig. 10: Feature importance for random forest model with minimal preprocessing

Feature engineering

To improve the performance beyond minimal preprocessing, we performed feature engineering. This involved creating 20 additional features by performing operations such as subtraction or division between existing features. Some new features are proportions of one variable over different periods (e.g. total opened account in the last 6 months/total opened account in the last 12 months). Others were the proportion between different types of variables (e.g. the number of secured accounts/the number of unsecured accounts). Using these new features, we constructed baseline models.

Baseline models

We then experimented with 4 different baseline models: decision tree, logistic regression, random forest, and XGBoost. These models were chosen for some of the following reasons:

Decision Tree	*Logistic Regression*	*Random Forest*	*XGBoost*
Simple and interpretable	Simple and interpretable	Can handle imbalanced data	Can handle missing data
Handles both numerical and categorical data	Handles both numerical and categorical data (with 1 hot encoding)	Handles both numerical and categorical data	Handles both numerical and categorical data (with 1 hot encoding)
Captures non-linear relationships	Probabilistic output aids decision making	Robust to overfitting	Robust to overfitting with built in regularization
Computationally efficient	Computationally efficient	Computationally efficient	Computationally efficient

Table 2: Benefits of baseline models

We split the data into train and test datasets. Feature engineering led to an improvement in the random forest model's test set's F1-score to 0.715, making a 0.02 increase over minimal preprocessing (Table 3).

Additionally, some of the newly engineered features such as *enq_L3m_divide_enq_L6m* (last 3 months' enquiries/last 6 months' enquiries) and *Age_Oldest_TL_Substract_Age_Newst_TL* (age of oldest opened account minus age of newest opened account), appeared in the top 10 most important features, indicating that feature engineering contributed to performance gains.

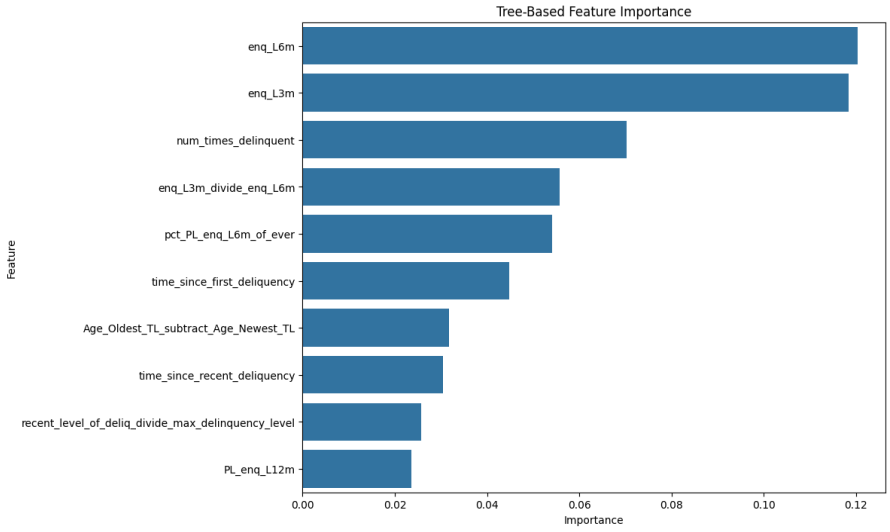

Fig. 11: Feature importance for random forest model with feature engineering

In addition, to address the data imbalance, we employed SMOTE (Synthetic Minority Over-sampling Technique) to upsample the minority class. Since SMOTE does not allow NaN, we dropped the features with NaN values. This yielded the F-1 score of 0.710 (Table 3) on the test set. However, SMOTE eventually performed poorer than baseline models that employed feature engineering, hence was removed as a preprocessing technique moving forward.

Across all baseline models' test sets' performances, we observe as expected, that due to the simplicity of its structure, the decision tree model underperformed compared to random forest by 0.012 in F-1 Score. Interestingly, logistic regression, which we initially expected to underperform due to the simplicity, performed nearly as well as random forest, achieving an F-1 score of 0.713. In conclusion, XGBoost outperformed all the models with a significantly higher F-1, 0.758.

The performance of each model is summarized in Table 3.

Feature Engineering	No	No	No	Yes	Yes	Yes	Yes
Model	Random Forest (minimal preprocess)	Random Forest (best preprocess)	Random Forest (SMOTE + drop features)	Logistic regression (best preprocess)	Decision tree (best preprocess)	Random Forest (best preprocess)	XGBoost (best preprocess)
F-1 score	0.696	0.706	0.710	0.713	0.703	0.715	**0.758**

Table 3: Models' performance

5. GTL Implementation and Results

To assess the potential of TabFM models for tabular data, we used a pre-trained TabFM model (Llama2-GTL) to the credit approval prediction task of TD Bank, consisting of predicting credit approval, without performing any prior fine-tuning.

We used the 7-billion (7B) and 13-billion (13B) parameter Llama2-GTL models for inferencing. Specifically, as shown in Figure 12, we used both the unquantized and quantized (8 bit) versions of the 7B GTL model. To run the 7B GTL unquantized model, we used an A100 GPU (40GB VRAM) from Google Colab Pro. The 7B 8 bit model could be run on Colab Pro or GCP instances (with T4 16 GB VRAM, L4 24GB VRAM GPUs).

However, for the 13B GTL model, we only used the 8 bit version (on similar T4/L4 GPUs) due to resource limitations. The 13B GTL unquantized model requires 53GB of VRAM, which exceeded our available local/GCP cloud resources.

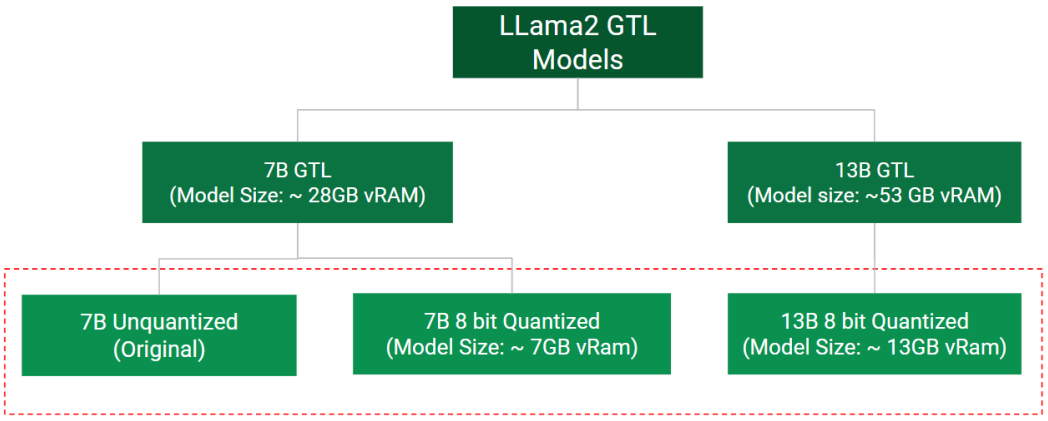

Figure 12

We tested the GTL models using two prompt templates from the paper [1]: **T-table and T-anony**. The key difference between the two approaches lies in the amount of information provided to the model through the prompt. In the T-table template, the model receives an initial prompt, feature descriptions, label description, in-context examples (training data), and the query. In contrast, the T-anony template excludes both the feature descriptions and the label description. Additionally, while the T-table template includes column/feature headers in the in-context examples, the T-anony template does not.

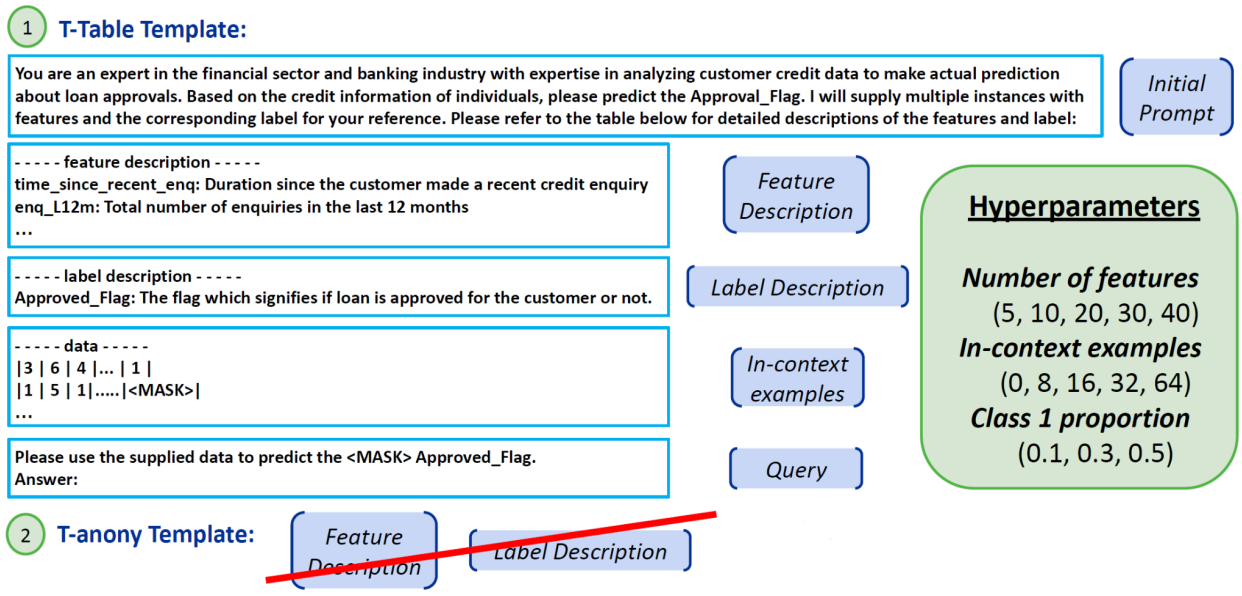

Figure 13

For each of the three models and two prompt templates, we analyzed the performance by fine-tuning the following hyperparameters: the number of features[3], the number of in-context training examples provided, and the proportion of Class 1 examples amongst them.

5.1. Analysis of Experiments & Key Observations

We used the F1 score as the primary evaluation metric due to the class imbalance in the dataset and because we are predicting discrete classes (0 or 1) rather than probabilities, which makes AUROC unsuitable for this task[4].

To assess the statistical significance of F1 score differences across hyperparameter variations, we applied Kruskal-Wallis and Dunn's tests[5][5]. The differences in the various experiments' F1 scores may at times be small, hence these tests help determine if the observed differences are statistically significant. The Kruskal-Wallis test is used to evaluate overall differences in F1 scores across multiple categorical variables (e.g., Prompt Type, Model, Sample Size). When Kruskal-Wallis indicates a significant result, Dunn's test is performed for more detailed pairwise comparisons to identify which specific level pairs (e.g., 7B vs 13B, 7B vs 7B 8 bit) drive the significant differences. This approach ensures a robust, quantitative analysis of how hyperparameters influence F1 score performance.

From this analysis we got the following key observations.

Key Observation 1: T-Table beats T-anony

We found that the T-table template outperforms the T-anony template, demonstrating that providing the model with more information leads to better F1 score performance.

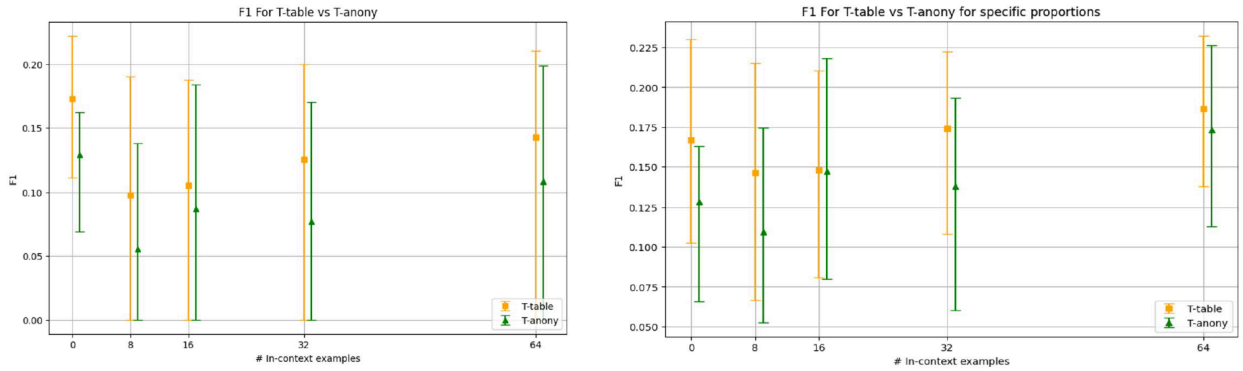

Figure 14

[3] The features were selected based on their feature importance.
[4] AUROC was used by the Microsoft paper
[5] They work for datasets that violate the normality assumption, like ours. We could not use the ANOVA nor Fisher's Exact Tests, which required that assumption to be valid.

Indeed, as shown in the candle plots in Figure 14, we observed that the T-table template generally outperforms T-anony. Furthermore, as shown below, statistical tests confirmed that these visual differences in F1 scores are statistically significant.

```
Kruskal-Wallis Test for Prompt Type:
  Statistic = 27.108945476376142, p-value = 1.923060099470278e-07
  Significant differences exist in Prompt Type. Performing Dunn's Test for post-hoc analysis...

Dunn's Test Results for Prompt Type:

              t_annony      t_table
t_annony  1.000000e+00  1.923060e-07
t_table   1.923060e-07  1.000000e+00
```

Figure 15

Key Observation 2: 13B-GTL-8 bit beats 7B-GTL-8 bit

When comparing the 13B and 7B quantized models, we found that the larger model outperforms the smaller one. This observation was initially noted in Table 4 and further confirmed through the statistical tests.

```
Kruskal-Wallis Test for Model:
  Statistic = 65.72920918072937, p-value = 5.334375413655906e-15
  Significant differences exist in Model. Performing Dunn's Test for post-hoc analysis...

Dunn's Test Results for Model:

                13B-GTL-8bit   7B-GTL-8bit  7B-GTL-unquant
13B-GTL-8bit    1.000000e+00  4.505261e-11    2.271530e-12
7B-GTL-8bit     4.505261e-11  1.000000e+00    1.000000e+00
7B-GTL-unquant  2.271530e-12  1.000000e+00    1.000000e+00
```

Figure 16

The Kruskal-Wallis test indicates that, in aggregate, F1 score differences due to model type are statistically significant. Furthermore, Dunn's test reveals that the performance of the 13B model is statistically different from both the unquantized and quantized versions of the 7B model.

Notably, for the 7B model, there is no statistically significant difference in F1 scores between the unquantized and quantized versions, suggesting that the quantized version could be used to save resources.

Key Observation 3: Higher the Class 1 proportion, better the F1 score

As shown in Figure 17 for the 13B GTL 8 bit model (the best model), a higher Class 1 proportion (i.e., more class-balanced in-context examples) leads to a better F1 score. Specifically, a balanced set of in-context examples (Class 1 proportion of 0.5) outperforms a proportion of 0.3, which in turn outperforms the case with the lowest Class 1 proportion, which has the most imbalance in the in-context examples.

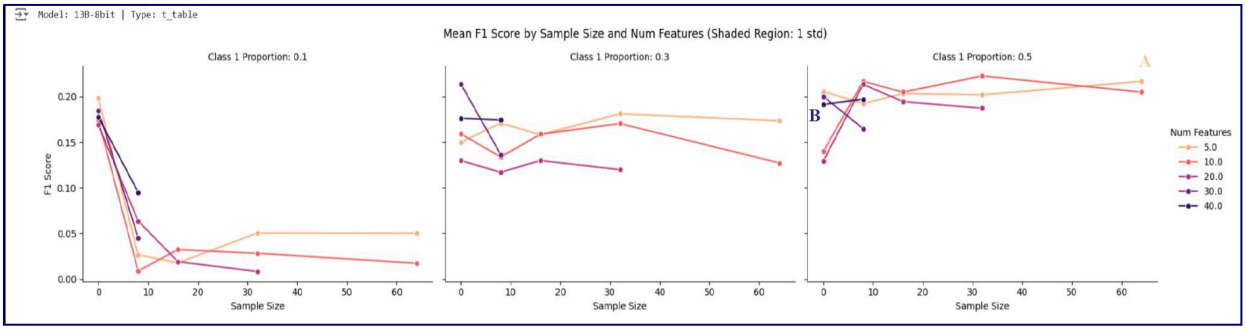

Figure 17

Although this difference does appear substantial visually, we still tested for its statistical significance, to get a robust conclusion.

```
Kruskal-Wallis Test for Class 1 Proportion:
  Statistic = 832.2405997476844, p-value = 1.910952785165962e-181
  Significant differences exist in Class 1 Proportion. Performing Dunn's Test for post-hoc analysis...

Dunn's Test Results for Class 1 Proportion:

              0.1           0.3           0.5
0.1   1.000000e+00  1.153112e-63  2.712222e-180
0.3    1.153112e-63  1.000000e+00   4.323151e-32
0.5  2.712222e-180  4.323151e-32   1.000000e+00
```

Figure 18

As demonstrated, the Kruskal-Wallis test confirms that F1 score variations based on Class 1 proportion are statistically significant. Moreover, Dunn's test indicates that these differences are significant across all pairwise comparisons of Class 1 proportions.

Key Observation 4: A large sample size with a small number of features is preferred over a small sample size with a large number of features.

One limitation of large language models (LLMs), such as Llama2, is the context length, which for Llama2 is restricted to 4096 tokens. This constraint requires balancing the number of features and the sample size to fit within the available context window. While both the number of features and sample size are statistically significant for predictive power (see Figure 19), the challenge lies in prioritizing which one to focus on given this constraint. In general, a larger

sample size with fewer features is preferred, as it provides more diverse data for the model to learn from, without overwhelming the model's context limit.

```
Kruskal-Wallis Test for Sample Size:
  Statistic = 44.03083093938333, p-value = 6.3218671299883786e-09
  Significant differences exist in Sample Size

Kruskal-Wallis Test for Num Features:
  Statistic = 34.27986845720759, p-value = 2.0942321379412995e-06
  Significant differences exist in Num Features
```

Figure 19

The preference for a larger sample size over a larger number of features is evident in Figure 17, particularly in the case of balanced in-context examples for the 13B GTL 8 bit model. Specifically, Point A, which represents a large sample size with a small number of features (64 samples and 5 features), achieves a higher F1 score than Point B, which represents a smaller sample size with a larger number of features (0 samples and 40 features).

6. GTL vs. Traditional Models Performance Comparison

To ensure a fair comparison, we applied the same limitations (eg. used the same number of features, sample sizes, and Class 1 proportions, to compare against a similar experiment run with a GTL model) that were imposed on the GTL models, onto the traditional models.

As shown on Table 4 - which shows the F1 scores obtained from averaging across different Number of Features and Class 1 sizes, the best GTL model (13B-GTL-8bit with T-table prompt type) loses to Logistic Regression but beats other traditional ML models (XGBoost, Random Forest, Decision Tree). However, it is important to note that the performance of the best GTL model remains relatively stable across different sample sizes, whereas Logistic Regression significantly declines in performance as the sample size increases.

F1 Score Means:

Prompt Type	Model	Sample Size	0.0	8.0	16.0	32.0	64.0	Row Mean
null	DecisionTree	NaN	0.043	0.079	0.119	0.134	0.094
	LogisticRegression	NaN	0.347	0.215	0.155	0.114	0.208
	RandomForest	NaN	0.036	0.053	0.076	0.093	0.065
	XGBoost	NaN	0.065	0.100	0.150	0.162	0.120
t_anony	13B-GTL-8bit	0.119	0.093	0.111	0.097	0.113	0.107
	7B-GTL-8bit	0.130	0.073	0.084	0.097	0.101	0.097
	7B-GTL-unquant	0.112	0.069	0.102	0.078	0.111	0.094
t_table	13B-GTL-8bit	0.173	0.131	0.124	0.130	0.132	0.138
	7B-GTL-8bit	0.146	0.084	0.099	0.114	0.143	0.117
	7B-GTL-unquant	0.202	0.096	0.095	0.108	0.124	0.125

Table 4

Conclusion

Without fine tuning for the new task[6] and using limited data[7] (a potential scenario that TD envisioned in the Problem Statement), both with and without feature descriptions, GTL models outperform some traditional ML models. While Logistic Regression appears to perform best overall, GTL models have the potential to surpass it by increasing the number of in-context examples and leveraging a larger model size. The quality of in-context examples provided to the model through the prompt matters and balanced examples would lead to better predictions and performance.

Ethical considerations:

We're currently working with a publicly available Kaggle dataset that contains no Personally Identifiable Information (PII). Similarly, if the proposed model is adopted at TD, data privacy compliance is ensured as we were told by our TD mentors that the same dataset schema is being used at TD and steps are taken to mask the PII data before project use. Moreover, through our experiments using class1 proportion as a hyperparameter during model inference, we identified that TabFM models too are sensitive to bias and for cases when the incontext samples provided were imbalanced, we noticed biased predictions. Therefore, care must be taken to ensure that when GTL is used, the in-context examples should be balanced in order to prevent bias amplification.

[6] TabFM/GTL models did not require training.

[7] TabFM/GTL (and traditional ML models) were limited to 4096 tokens worth of "training/in-context examples".

7. Contribution

- Yu Zheng Lim (yl5451): EDA, literature review, report writing, statistical analysis of results, GTL model experiment
- Ryu Sonoda (rs4493): EDA, baseline modeling, GTL model experiment, analysis of result, report writing
- Atharva Kulkarni (ak5070): EDA, Literature Review, GTL model experiment, Poster and Executive Summary
- Nicolas Cogorno (nac2216): EDA, GTL model experiment, analysis of result, report writing, Executive Summary
- Minh Dang (nd2802): Appendix: Ensemble, GPT-4 model experiment
- Varun Agarwal (va2515): Team Captain: set up meetings and manage progress. EDA and report writing, GTL model experiments, GCP infra setup for the team

Project code and results:
https://github.com/engie4800/dsi-capstone-fall-2024-team-td-ltm

References

[1] Xumeng Wen, Han Zhang, Shun Zheng, Wei Xu, Jiang Bian. 2023. From Supervised to Generative: A Novel Paradigm for Tabular Deep Learning with Large Language Models. In KDD.

[2] Tianqi Chen and Carlos Guestrin. 2016.XGBoost: A scalable tree boosting system. In KDD.

[3] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021.Revisiting deep learning models for tabular data. In NeurIPS.

[4] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023.TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. In ICLR.

[5] Zach Bobbitt. 2020. Dunn's Test for Multiple Comparisons, web blog, accessed Dec 2024. <https://www.statology.org/dunns-test/>

Appendix

Below are some additional ideas we explored during the project.

A. Ensemble

When we finished running all the models, we implemented and compared four different ensemble methods to aggregate the results:

- Majority Voting: select the most common prediction among models
- Weighted Voting: use F-1 scores to assign greater influence to better models in the process
- Confidence-based: consider only predictions from models with F-1 score > a threshold (0.7 in this case)
- Dynamic-weighted voting: adjust model weights dynamically based on a rolling average of recent F-1 scores (most recent 3 in this case), putting more influence for well-performing models over time
- Stacking: combine predictions using a meta-classifier (Logistic Regression in this case) trained on model outputs, and capture relationships between predictions to improve the performance

We performed ensembles by prompt types, as we care about the F-1 mean across all sample sizes.

For ensembled GTL, the results in general were positive. All F-1 scores when ensembled were improved over single model runs, though some instances fell short of the exceptional 8-sample-size Logistic Regression run:

	t-anony	t-table
Majority Voting	0.396087	0.281389
Weighted Voting	0.149239	0.229471
Stacking	0.779458	0.800872
Confidence-based	0.396087	0.281389
Dynamic-weighted	0.386992	0.288887

Observations:

- Stacking outperformed other methods since the meta-classifier corrected individual model biases and leverage complementary strengths
- Weighted voting underperformed, likely due to its assumption of linear proportionality between F-1 scores and model reliability, which may fail when models struggle with specific sample sizes or class distributions.

Actionable Conclusions:

- Adopt Stacking as the primary ensemble strategy: this approach should be prioritized after finishing all 100 GTL runs to combine and optimize final predictions
- Increase sample size in test runs: could reduce bias and improve the performance of all ensemble methods, as smaller sample sizes lead to inconsistencies or NaN values in metrics

Challenges: Some sample sizes yielded some NaN (not a number) values, likely caused by:

- Undefined precision or recall when no prediction were made for certain classes
- Omission of minority class in doing majority votes
- Ignorance of low-confidence predictions in confidence-based method
- Challenges in generalizing underrepresented class
- Averaging and threshold effects, including window sizes and minimum periods

When considering ensemble methods between traditional ML and GTL models, the lack of identical grouping categories (such as prompt type for GTL or tuning for traditional,) limits their compatibility. Exploring alignment strategies to facilitate effective across these approaches could be of future research. Additionally, fine-tuning hyperparameters such as confidence thresholds, window sizes, voting strategies, and using larger or more diverse datasets could improve robustness.

B. Running GPT-4 models from OpenAI

On top of the Llama-2 GTL models on HuggingFace, we also performed tests on GPT-4 models to see if the performance is better, using **APIs**.

For the first few runs, I went for **GPT-4o**, the flagship model, but the total cost was getting exponentially bigger after the first 2 sample sizes ran on t-table. We had more than 700,000,000 input tokens in total; if we used the regular pricing of $2.50 for 1M tokens, it would cost us $1,750 just for input only.

With a constraint in resources, GPT-4o wasn't ideal. As a result, we moved towards **GPT-4o-mini**, which was 17 times cheaper ($0.15 only). To be more cost-efficient, we also utilized **Batch API inference**, which allows us to create large batches of API requests for asynchronous processing for a 50% discount. The only hard thing about this process is that you have to follow OpenAI's batch formatting, otherwise, it's going to reject your request (which still costs you money). Eventually, the total cost was reduced by 94% (to around $105.)

Evaluation-wise, GPT-4o-mini's F-1 score was better than that of the GTL models with t-anony, except for the 8 sample size of the same template compared to Logistic Regression:

Prompt Type	Model	0.0	8.0	16.0	32.0	64.0	Row Mean
null	DecisionTree	NaN	0.043491	0.079389	0.118867	0.133856	0.093901
	LogisticRegression	NaN	0.347454	0.215288	0.155047	0.113595	0.207846
	RandomForest	NaN	0.036094	0.053404	0.075824	0.092732	0.064514
	XGBoost	NaN	0.065304	0.100108	0.150293	0.162647	0.119588
t_annony	13B-GTL-8bit	0.119103	0.092935	0.110993	0.097328	0.112854	0.106643
	7B-GTL-8bit	0.130229	0.072619	0.083966	0.097300	0.100534	0.096930
	7B-GTL-unquant	0.111618	0.068523	0.120932	0.078351	0.110749	0.098035
	GPT4	0.228230	0.216957	0.224096	0.221100	0.212759	0.220628
t_table	13B-GTL-8bit	0.173315	0.130500	0.124377	0.130009	0.131732	0.137987
	7B-GTL-8bit	0.145590	0.084342	0.098623	0.114485	0.142783	0.117165
	7B-GTL-unquant	0.201892	0.095752	0.094629	0.108355	0.123731	0.124872
	GPT4	0.202756	0.202212	0.194154	0.178120	0.154475	0.186343

GTL, by its name, and claimed by the paper, should be more suitable for tabular data than GPT. However, there might be some factors that lead to this result:

- GPT models are large-scale with optimized architectures and fine-tuned reasoning and might be better at understanding data and converting them for prompt engineering
- GPT models generalize better with their extensive pre-training, leading to better results on different domains/contexts and unseen tabular relationships
- GPT models are designed for zero-shot or few-shot learning, making them effective for minimal training data tasks (which is our main goal in this project)

TD might not want to use APIs with all that sensitive information from the customers – the fact that we have to use external datasets on Kaggle for this project and not the real data from TD – is concrete proof. But as the regulations on AI evolve with how that data can be shared between different organizations, this would be worth trying in our opinion, even a few years from now, with all the advantages of the GPT model's architecture.

When it comes to processing time, running API requests sequentially without a batch is not only expensive but also very slow. When the first batch was run sequentially, there were around 11,000 requests, taking more than 2.5 hours. To reduce the running time, multiple threads were run simultaneously within a processor (**multithreading**). As a result, the remaining 200,000 requests of the batch only took 2 hours in total. Compared to the GTL running time of 90 seconds/100 predictions, this multithreading process **is 6 to 18 times** faster per 100 predictions. Since this wasn't integrated into the GTL runs, this could be applied in future tests.