Содержание

1 Introduction

1.1 Event sequences

Event sequences [1] refer to a series of events that occur in a particular order, often with some sort of causal or temporal relationship between them. Event sequences can be found in various fields, including computer science, natural language processing, and linguistics. In the context of computer science, event sequences are commonly used in the analysis of log data. Log data can contain a large amount of information about system behavior, user actions, and other events. By analyzing event sequences within this log data, it is possible to gain insights into patterns of behavior, identify potential issues or errors, and optimize system performance. Examples of such types of data are log entries, Internet of Things telemetry, industrial maintenance, user behavior, travel patterns, medical history, transactional data, click-through rates, and other industrial and financial event sequences [2]. One technique for analyzing event sequences is machine learning [3]. Machine learning algorithms can be trained on large datasets of event sequences to identify patterns and make predictions about future events. For example, machine learning algorithms can be used to predict which products a customer is likely to purchase based on their past purchasing behavior. Event sequences are a powerful tool for analyzing behavior and understanding patterns in a wide range of fields. By analyzing event sequences, it is possible to gain insights into system behavior, language usage, and other complex phenomena.

1.2 Text as a universal task layer

Text is a universal task layer that serves as a fundamental building block for human communication and information exchange. From the earliest forms of writing to the latest digital technologies, text has been an essential tool for organizing and conveying information in a wide range of contexts, including education, business, science, and entertainment. At its core, text is a means of encoding information into a structured format that can be easily shared and understood. Whether it's a simple message written on a piece of paper or a

complex scientific report, text provides a universal framework for organizing and transmitting information across diverse audiences and platforms. One of the key benefits of text is its accessibility. Unlike other forms of communication, such as spoken language or visual media, text can be easily translated and shared across cultures and languages. This makes text an ideal tool for promoting cross-cultural understanding and collaboration, particularly in globalized contexts where diverse perspectives and communication styles are essential. Text is also a highly versatile tool that can be adapted to a wide range of tasks and contexts. From writing emails and reports to programming and data analysis, text provides a powerful medium for organizing and manipulating information in a structured and meaningful way. Whether it's a simple list or a complex database, text can be used to represent information in a way that is easy to understand and manipulate in the form of text interfaces.

1.3 Text interfaces

Text interfaces are a type of user interface that allows users to interact with computer systems through text commands. These interfaces have been used since the early days of computing and continue to be popular today due to their speed, efficiency, and flexibility. At their most basic level, text interfaces consist of a prompt, a cursor, and a text field where users can enter commands. Once a command is entered, the system processes it and returns a text-based response. This response can include information, feedback, or an error message, depending on the command and the system's capabilities. Text interfaces can be found in a variety of applications, including operating systems, programming environments, and software development tools. They are particularly popular among developers and power users who value the speed and precision of text-based commands. Some examples of text interfaces include the Windows Command Prompt, the Unix shell, and the Python interpreter. In recent years, there has been a renewed interest in text interfaces as more developers and users seek out lightweight and flexible tools. Many popular software applications, including code editors and development environments, now include built-in text interfaces that allow users to perform

advanced tasks and automate repetitive workflows. However, text interfaces have traditionally been limited in their ability to understand and respond to human language naturally. Large language models [4] have changed this by enabling computers to understand and generate human-like text. These models are trained on massive amounts of data and can learn to generate text that is indistinguishable from human writing. This has made it possible to create text interfaces that can understand and respond to natural language queries in a more natural and human-like way.

1.4 Large language models as text interfaces

One of the most popular applications of large language models in text interfaces is in the creation of chatbots. Chatbots are computer programs that can simulate conversation with users through text or voice. They are often used by businesses and organizations to automate customer service and support functions. Chatbots can answer frequently asked questions, provide information about products and services, and even help users troubleshoot technical issues. Large language models have made chatbots more effective by enabling them to understand and respond to user queries in a more natural and human-like way. This has led to an increase in user engagement and satisfaction with chatbot interactions. Chatbots powered by large language models can also learn from previous interactions with users and improve their responses over time. Another application of large language models in text interfaces is in virtual assistants. Virtual assistants are computer programs that can understand and respond to voice commands. They are often used to perform tasks such as setting reminders, making appointments, and controlling smart home devices. Large language models have made virtual assistants more effective by enabling them to understand and respond to natural language voice commands in a more human-like way. This has led to an increase in the adoption of virtual assistants and their use in a wide range of applications.

1.5 Application of text interfaces

The application of text interfaces doesn't stop in the text domain - Visual Language Models (VLMs) have become increasingly popular in recent years, as they offer a more accessible way for users to interact with and control these models. This technology has numerous applications across a range of industries, including entertainment, gaming, marketing, and education. One of the primary benefits of text interfaces for visual language models is that they allow users to communicate complex ideas and concepts in a more natural and intuitive way. For example, in the "Segment Anything"paper [5], authors proposed using text to define segmentation masks. Another example is the Imagen work [6], where images are generated from text. This is particularly important in fields such as design, where visual communication is a crucial part of the creative process. By using a text interface to control a visual language model, designers can easily experiment with different visual elements and layouts without spending hours manually creating and tweaking each individual design. For example, using VLMs, it is possible to blend text and images or edit images using text [7].Another key benefit of text interfaces for visual language models is that they allow users to quickly and easily generate large amounts of content. This is particularly useful for marketers who often need to create multiple versions of the same message for different audiences and platforms. By using a text interface to generate visual content, marketers can easily create and customize images and videos to fit their specific needs without spending time and resources on manual design work.

1.6 Aim of research

The success of text interfaces in the text and visual domains motivates us to explore the possibilities of constructing text interfaces for different modalities. The main purpose of this research is to investigate the possibility of creating text interfaces for a new modality. For this new modality, we have chosen sequential data. Event sequences are a pure example of multimodal data because they contain information from different sources and have different natures, such as age, education, and amount. Additionally, this type of data has variable lengths,

which allows us to investigate how text interfaces can work on other sequential modalities besides text. Our research has several sub-goals:

- **Creation of a proper representation for event sequences**: To pass data to text interfaces, we first need to preprocess it. Therefore, we have to construct an encoder model and conduct experiments to determine which representation works best because ultimately, a better representation will yield better results on text interfaces.
- **Construction of a question-answering format task on sequential data**: To examine whether it is possible to construct a text interface for event sequences, we need to figure out how to transform the task into a question-answering form and check whether text interfaces can work in this setting.
- **Conducting experiments in several tasks**: One promising ability of text interfaces is their ability to solve multiple tasks simultaneously. Therefore, we are interested in conducting experiments to explore the possibilities of text interfaces in solving various tasks on event sequences.

2 Related works

2.1 Event sequence processing

There are several ways to construct event sequences in an unsupervised manner for representation over event sequences. The first is to apply masked language modelling [8] where some events are masked, and the model learns to reconstruct the missing tokens. In this task, a certain percentage of tokens is masked, and the objective is to reconstruct them. The second task is token replacement detection [9]. The essence of this task is to replace some events with random ones, and the model's goal is to detect which token was replaced. This task is commonly considered more efficient because the gradient passes through all events, not only the masked ones. Additionally, there is a method called Contrastive Learning for event sequences where contrastive learning is applied. The speciality of this method lies in how positive examples are constructed. Positive examples are constructed by splitting the sequence into several subsequences. The motivation behind this approach is that the representation of the object doesn't change over time. Therefore, the embedding of different subsequences should be the same. The full survey can be found in [10]. The main idea of this paper is that whether you use a different pretraining scheme for better performance on the final task, one should use target information during the pretraining stage.

2.2 Feature processing

The first necessary step in data processing is to transform the data into numerical values. To transform categorical features, the classical approach is to apply one-hot encoding, where each category is transformed into a vector representing ones in corresponding positions and zeros elsewhere. However, this approach becomes inefficient when dealing with a large number of features, since the number of features increases linearly with the number of categories. An alternative approach is target encoding, where categorical features are replaced with probabilities of the target class. The most popular and universal approach is the embedding approach, where each categorical value is mapped into a learnable

embedding and passed further into the model. When dealing with numerical values, there are several approaches to process them. The first is to apply logarithmic transformation, normalize them, or apply value clipping. As researchers have found that transformers perform poorly on numerical data, they propose applying discretization and transforming numerical features into categorical features. This approach handles outliers effectively and works well with numbers. However, this approach has a limitation, as discretization reduces the ordering between numbers. In the work [11], ordering is preserved by introducing a piecewise linear encoding, where an embedding for a new value is constructed by summing the embeddings for all previous values. Additionally, in this paper, they propose a new scheme for discretization. In the classical discretization approach, bins are created using quantiles of the numerical features. In [11], they propose constructing bins using a tree. This approach incorporates target information into the embeddings and improves the quality of algorithms trained using this type of embedding. Once the representation for each feature is constructed, it is necessary to combine them. The standard scheme is to concatenate them. However, this can harm performance as it doesn't consider the inter-feature relationships in the model. In the work [12], the embeddings of categorical features are additionally processed with transformers because the context is meaningful. This architecture was further modified in the work [13] where multilayer perceptrons were replaced with gating multilayer perceptrons. The same approach is applied in [14] where feature embeddings are processed and only then concatenated into transaction embeddings.

2.3 Architectures

There are different architectures that can process event sequences. In the early stages, recurrent neural networks (RNNs) were used to process event sequences. For example, in the work [15], recurrent neural networks were applied. Another relevant work is [16]. However, with the rise of transformer models [17], they have become a standard for sequence processing, and event sequences are no exception. In [18], the author adapted ResNets and Multilayer perceptrons. They have also shown that ResNets are better at imitating boosting trees than

MLPs. A closely related topic to event sequences is time series. Time series can be considered as event sequences with less data, focusing primarily on values. The main problem when applying transformers to event sequences is that attention scales quadratically with the number of inputs, which becomes a significant issue for long sequences. One idea to address this problem is to process the data sequentially by splitting it into accessible parts. However, this approach does not work well for very long sequences because the model forgets the context of previous windows. A solution to this problem was proposed in [19], where intermediate outputs of the previous model are reused to retain information from previous stages. Another idea is to use sparse attention mechanisms. In this case, the complexity of attention is not $O(n^2)$, but rather, $O(n \log n)$ or even $O(n)$. For linear attention, a mechanism called linear attention is proposed. In this situation, linear attention is split into scaled products, resulting in linear complexity [20]. Another technique is to selectively choose only a few elements for which attention is calculated. Examples of such works include Informer [21]. In [22], the authors were able to reduce complexity to linear by randomly sampling a constant number of Fourier frequencies.

2.4 Large language models

Large language models have emerged in natural language processing and have become the de facto standard. They are trained on large text corpora in an unsupervised manner using Masked Language Modelling, Causal Language Modelling, and Prefix Language Modelling. Through these tasks, models are able to solve a variety of tasks. Examples of such models include GPT-3 [4], BART [23], T5 [24], UL2 [25], PaLM [26], and BERT [8]. These models have demonstrated incredible capabilities in tasks such as reasoning, semantic parsing, sentence classification, text generation, and question answering.

When applied to event sequences, language models have shown good performance in data-to-text tasks. A survey of different approaches is considered in [27]. In [28], language models are applied to transform tables into text. In [**?**], it is shown that language models can solve tabular problems better than logistic

regression and gradient boosting. The same is demonstrated in [29]. In [30], it is shown that language modelling can also generate realistic tables. The main feature of this type of model is that they can easily be adapted to conditional generation and realistically imitate missing data in the table. In [31], language modelling is used to solve semantic reasoning problems. The idea of semantic reasoning is to transform a natural language query into a structured query language that can be executed. Language models efficiently solve this task in [31, 32]. In [33], semantic parsing is applied to simplify the original task. In [34], a joint model is created that can map data to text and vice versa.

2.5 Prompt engineering

The foundation for text interfaces is provided by the prompt engineering technique. Creating a large language model takes a significant amount of time. Therefore, fine-tuning on a new task will also be time-consuming. However, large language models are adept at recognizing patterns and working with text. In [4], it has been observed that big models are capable of solving tasks in a zero-/few-shot manner. This concept becomes interesting because it is not necessary to train the model extensively; rather, the focus is on constructing prompts correctly. A survey of prompt engineering techniques is provided in [35].

The idea behind prompt engineering is as follows: large language models are powerful, and by constructing the request properly, the model can provide an answer. The request is constructed in the following form: [Input] text [Output]. The goal of the model is to accurately fill in the output. There are several considerations regarding how to map the output of the model to the label space.

There are several techniques for prompt engineering, depending on the extent to which the model is trained. The first approach is the classical approach, where a pre-trained language model is simply fine-tuned for the downstream task. The second approach is Tuning-free Prompting, where no parameters are trained, and a template is manually selected. The third approach is Fixed-LM Prompt Tuning, where the language model is frozen, but additional training prompts are added to the model. The fourth approach is Fixed-prompt LM Tuning, where a

fixed prompt is chosen, and the language model is trained to solve the downstream task. The last approach is Prompt+LM Fine-tuning, where both the language model and prompt parameters are trainable. This approach allows for better performance, but may be prone to overfitting and have less impact.

2.6 General purposes interfaces

The concept of text interfaces is prevalent in the text domain. The majority of text tasks involve both questions and answers in text form. In [24] and [36], the T5 model was able to solve over 2,000 tasks with comparable performance. Next, the concept of interfaces was extended to the multimodal domain, including text and images. Tasks such as object detection and image reasoning can be tackled using text interfaces. In the work by [37], the model proposed the use of text interfaces for image and audio modalities. In [38], the authors suggested combining different pretrained models to solve tasks in a zero-shot manner using templates. In the work by [39], the authors successfully combined a frozen language model with a visual encoder to solve tasks in both the image and text domains. Additionally, GPT-4 also offers capabilities for multimodal dialogue in the work by [40].

In [41], large language models were applied as application programming interfaces, allowing them to perform tasks such as sending emails, using calculators, and solving mathematical problems. A similar approach was proposed in [42], where text interfaces were applied to recommendation tasks. In this domain, the model can provide recommendations based on dialogues with the user, generate summaries of reviews, make sequential recommendations, provide explanations for model predictions, ask whether the user likes the recommended film, recommend films from a list, and predict the user's rating of a film. Furthermore, this model can handle tasks in a zero-/few-shot manner. mb

3 Methodology

3.1 Text interfaces

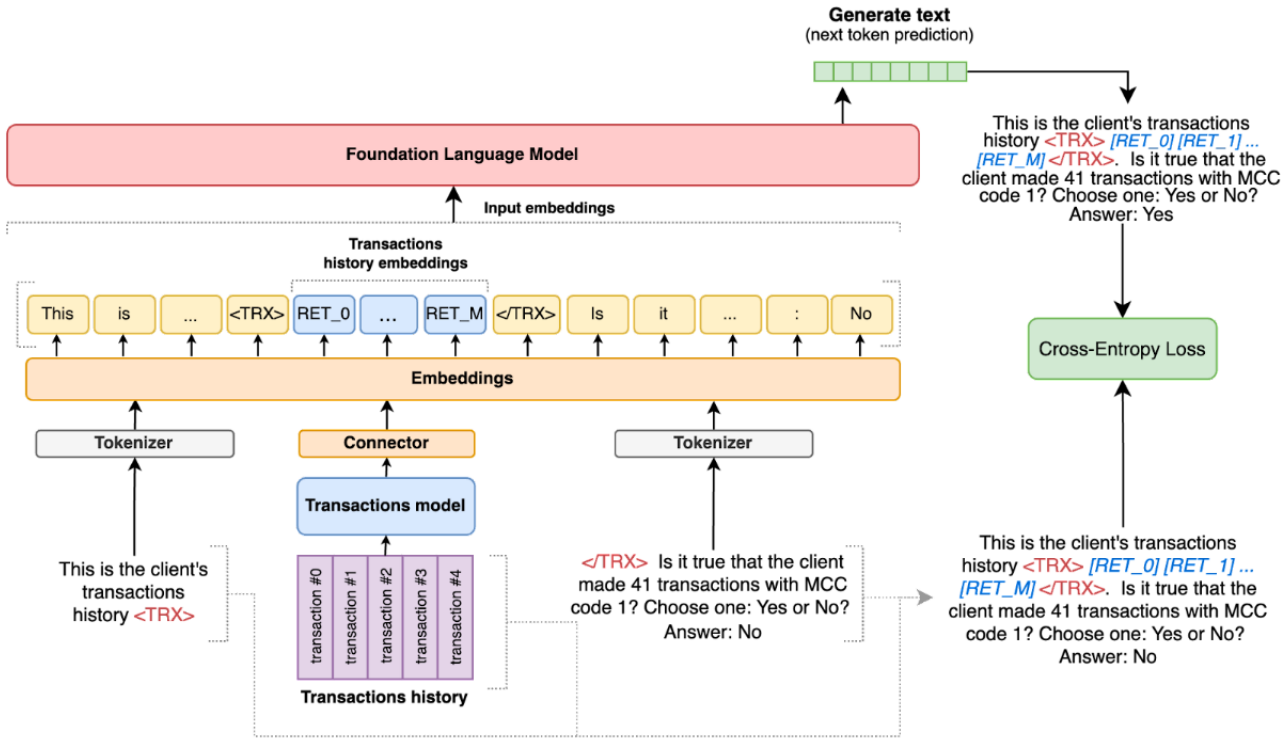


Рис. 1: A text interface built for event sequences. Here the name representation model is replaced by a transactions model

In our work, we define text interfaces as a large language model that takes the input prompt $\mathcal{Q}$, encoded data $\mathcal{E}(D)$, and produces an output answer $\mathcal{A}$. A text interface consists of two submodels: a representation model for data $\mathcal{E}$ and a text model $\mathcal{T}$. The representation model processes input D to produce a sequence of embeddings $\mathcal{E}(D)$. The text model $\mathcal{T}$ accepts both the text prompt $\mathcal{Q}$ and the encoded data $\mathcal{E}(D)$. To learn text interfaces, we train our model to solve language modeling using cross-entropy loss on the next token prediction task. Mathematically, it can be written as:

$$\mathcal{L}_{AR} = -\sum_{i=1}^{n} \log p(\mathbf{y}_i | \mathbf{y}_{<i}, \mathbf{x}; \theta), \tag{3.1}$$

where $\mathbf{y}< i$ denotes the sequence of words preceding the i-th word, $\mathbf{x}$ represents

encoded data, and $p(y_i|\mathbf{y}< i; \mathbf{x}, \theta)$ is the probability assigned by the model to the i-th word given the preceding words, encoded data, and θ represents the model parameters.

3.2 Dataset

Training text interfaces requires that we have a text prompt for every data in the form of a question and answer. However, most datasets do not have corresponding text input. Therefore, we propose a procedure on how to generate text input for data. The idea is to transform the mathematical task formulation into a natural question and answer format. For example, instead of asking "When will the next event arise?"the text interface should take this as an input and produce the text answer: "In 10 days,"instead of just typing "10". We propose two types of questions: predictive and extractive.

3.2.1 Predictive tasks

Predictive tasks are responsible for predicting the future based on data. For example, we can predict the time of the next event. Therefore, they are of primary interest to us. To create a task, we utilize the following idea: every event sequence contains features that describe each event, such as the type of event. So, we propose predicting the next feature event value. Additionally, we propose predicting the aggregated value for the next series of events, such as predicting the number of events in the next 30 days.

3.2.2 Extractive tasks

Extractive tasks are tasks in which targets can be directly extracted from the data. For example, calculating the average time difference between transactions. Here, we propose the same scheme for task construction - we train the model to predict feature values or aggregates over the provided data, such as the most frequent value or the least frequent value. The motivation for this task is that it can be used for pretraining for a certain type of task and then fine-tuned for

similar predictive tasks. For example, we can fine-tune the model to predict the most frequent type of event and then apply it to predict the next event type.

3.2.3 Question

To make the text interface work with data, we need to reformulate the task in text form. For this purpose, for every task, we propose a text prompt that will be passed as input to the text interface, and an answer prompt that the model should produce.

The procedure for constructing the text input prompt is as follows. First, we prepend the following prefix: "This is the transaction history of the client."This text is intended to notify the model that it will receive the encoded event sequence next. Before passing the encoded sequence, we wrap it using new tokens «trx>"and «/trx>"to indicate the beginning and end of the transaction history, respectively. It is common practice to define start and end tokens when introducing a new modality. After that, we append the task-specific question. There are three types of questions:

- **Binary**: In the binary setting, the question takes the form of a general question, such as "Will the next event happen in the next 10 days?"The corresponding answer should be either "Yes"or "No."

- **Multiple-choice**: In the multiple-choice setting, we construct the question as follows. First, we ask the model an open-ended question, which usually has multiple answer options. To provide guidance to the model, we include the text "OPTIONS:"followed by several variants that can help the model answer the question. If there is no correct answer among the options provided, the model should output "neither."

- **Open-ended**: In the open-ended setting, we simply ask the model a question without any hints, and the answer can be in any form.

3.3 Task construction details

The construction of the question also depends on the type of task: numeric, where the corresponding output is a numerical number, or categorical, where the corresponding output is a category.

3.3.1 Binary

For binary numeric tasks, we considered the following approach for task construction. We construct binary tasks as predictions of whether the value will be greater or smaller than a given threshold. For example, "Will the feature value of the next event be greater than the median value?"We chose the median value because it allows us to create a balanced binary task.

For binary categorical tasks, we considered the following approach for task construction. Since categorical values have a limited set of values, we decided to use a subset of possible values and construct the task as a check to see if our prediction falls within the chosen set. For example, "Will the next transaction type be in the following group of classes?"

3.3.2 Multiple-choice

For numerical multiple-choice tasks, we need to provide several options. However, text models are not very good at predicting numerical values. Therefore, we decided to apply the following approach. We discretize the numerical feature into a set of non-intersecting intervals. For example, we can split the numerical feature into intervals: [0, 1000], [1000, 2000], [2000, 3000]. In this case, the task of predicting the exact number reduces to predicting the interval in which the number lies, which becomes a categorical task. We can then easily provide the options for it.

For categorical multiple-choice tasks, it is much easier to construct the options. The idea is to sample some classes and check if the answer lies among them. Otherwise, we return "Neither".

3.3.3 Open-ended

For open-ended tasks, the task is constructed by simply evaluating how close the predicted value is to the target value. In the case of a categorical feature, we require an exact match between the predicted and target values. For numerical answers, we measure the mean absolute error.

3.3.4 Mapping answer to label

Since the answer from our model is in text form, we need to process the text output to obtain predicted labels. Binary tasks are a simple case. Here, we only need to determine whether the answer is positive or negative. This can be achieved by mapping positive words, such as "Yes,True,Positive,"etc., to the positive class, and negative words, such as "No,False,Negative,"etc., to the negative class.

For multiple-choice tasks, where we have a predefined set of options, we can calculate the probabilities for each option and choose the one with the highest probability as the predicted label.

In the case of open-ended categorical tasks, the prediction needs to match the answer exactly. For open-ended numerical tasks, we convert the produced string into a numeric value. If conversion is not possible, we assign a value of -100.

3.3.5 Prompt Augmentation

Using the same question may not be sufficient to make a language model universal. The model can overfit to a specific pattern and provide random predictions for semantically similar questions that are syntactically different. Therefore, in our experiments, we created various question variations. We developed four different prefixes for the input definition and five postfixes for the question, resulting in 20 different question variations.

3.4 Creating representation for sequential data

To pass sequential data to embedding, we can utilize text interfaces. The basic idea is to transform the data into a text format. For example, "Event №1

happened at 12 am on December 21, 2022, with the following feature values. Event №2 happened at 1 pm on December 21, 2022."However, this representation is both too sparse and too long. Modern architectures, such as transformers, struggle to handle sequences with thousands of events. Therefore, we propose using an embedding formulation.

The concept is to transform the sequence of events into an embedding representation. There are two possible options: a one-to-one transformation, where each event has an independent embedding, or mapping the entire sequence into one or several embeddings. In our research, we focus on the one-to-one mapping approach. To construct the embedding for each event, we employ the following scheme: for numerical features, we apply a linear layer to map them to a higher-dimensional space, while for categorical features, we directly map them to embeddings. For sequence features that occur only once throughout the sequence, we map them to embeddings and concatenate them with the embeddings of all events. When we have embeddings for each feature independently, we concatenate them to construct the embedding of the entire sequence. The dimension of the embeddings is chosen empirically.

However, these raw embeddings lack connections between each other and do not produce meaningful representations. To address this, we train an additional model on top of this representation to consider the relationships between events. There are different methods available for building this model.

3.4.1 CoLES pretraining

CoLES is a technique used to generate representations for event sequences in a self-supervised manner. The key idea behind this approach is to use subsequences from the same user as positive examples and subsequences from other users as negatives. The motivation behind this is that the embedding of a user shouldn't change significantly over time. The CoLES loss is defined as follows:

$$\mathcal{L}_{uv}(M) = Y_{uv} d_M(u,v)^2 + (1 - Y_{uv}) \max\left\{0, \rho - d_M(u,v)\right\}^2, \tag{3.2}$$

where $d_M(u, v) = d(c_u, c_v)$ is the distance between embeddings of the pair (u, v), Y_{uv} is a binary variable identifying whether the pair (u, v) is positive, and ρ is the soft minimal margin between dissimilar objects.

3.4.2 CPC pretraining

Contrastive Predictive Coding (CPC) is a technique used for unsupervised learning of representations for sequential data, such as speech signals or natural language text. CPC takes an information-theoretic approach to feature learning, aiming to maximize the mutual information between a context window (a sequence of input features) and a predicted target feature. The basic idea of CPC is to learn an encoding function that maps a sequence of input features into a fixed-dimensional latent representation, and a prediction function that takes a pair of context window features as input and predicts the target feature using the corresponding latent representations. The trade-off between the encoding and prediction functions results in a compact and informative feature representation that captures relevant information about the input sequence.

The CPC loss function is defined as the contrastive cross-entropy between the true target feature and the predicted target feature, computed over a set of negative samples drawn from a noise distribution. The negative samples serve as distractors that encourage the model to learn meaningful patterns in the input data. The CPC loss function can be expressed as:

$$L_{CPC} = -\log \frac{\exp(z_{\tau+1}^{\top} g_\tau(z_{\tau-L:\tau}|\theta_E))}{\sum_{k=1}^{K+1} \exp(z_k^{\top} g_\tau(z_{\tau-L:\tau}|\theta_E))} \tag{3.3}$$

where $z_{\tau+1}$ is the true target feature, $z_{\tau-L:\tau}$ is the context window, and g_τ is the prediction function. K is the number of negative samples, and θ_E are the parameters of the encoding function.

3.4.3 Next event prediction

Another method of pretraining is next event prediction. This approach is similar to the task of causal language modeling, where the goal is to predict the

next event based on the event history. In classical language modeling, the task involves classifying the next token. In our case, we want to predict several features, including some that are continuous. Therefore, we propose the following scheme: for categorical features, we use cross-entropy loss, and for numerical features, we use mean absolute error (MAE) loss. The final loss is the sum of the losses for each feature prediction.

3.5 Connector

To connect the output of the representation model with the input of the language model, we need to ensure that their dimensionalities match. To achieve this, we propose using a linear layer to adjust the dimensionality accordingly.

3.6 Training regimes

For training, we consider three regimes: (1) training the entire model, (2) training the language model and connector, and (3) training the representation model and connector. All these regimes provide value for our experiments. Fine-tuning the entire model allows us to explore the limits of the approach. Fine-tuning the language model and connector demonstrates the potential to utilize a pretrained block for the representation. Fine-tuning transactional models showcases the possibility of using any language model to construct a text interface.

3.7 Metrics

For evaluating the performance of the models, we use three metrics: AUC for imbalanced classification problems, Accuracy for balanced classification problems, and MAE for regression problems.

3.7.1 AUC

AUC (Area Under the Curve) is commonly used in machine learning to evaluate the performance of binary classification models. It measures the area under the Receiver Operating Characteristic (ROC) curve, which represents the

trade-off between the true positive rate and the false positive rate at different classification thresholds. The AUC value ranges from 0 to 1, with a higher value indicating better performance of the classification model. The AUC formula is defined as follows:

AUC

$$AUC = \int_0^1 TPR(FPR^{-1}(t))dt,$$

where TPR is the true positive rate, FPR is the false positive rate, and $FPR^{-1}(t)$ is the inverse function of the false positive rate.

3.7.2 Accuracy

Accuracy is the ratio of correctly predicted data points to the total number of data points. It provides an overall measure of a model's performance. The formula for Accuracy is:

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN},$$

where TP represents the number of true positives, TN represents the number of true negatives, FP represents the number of false positives, and FN represents the number of false negatives.

3.7.3 Mean Absolute Error (MAE)

MAE is a commonly used metric in regression analysis that measures the average absolute difference between the predicted and actual values. It is calculated by taking the absolute value of the residuals and dividing by the number of observations:

Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

where y_i is the actual value of the dependent variable for the i-th observation, $\hat{y}_i$ is the predicted value of the dependent variable for the i-th observation, and n is the total number of observations.

4 Numerical experiments

4.1 Data

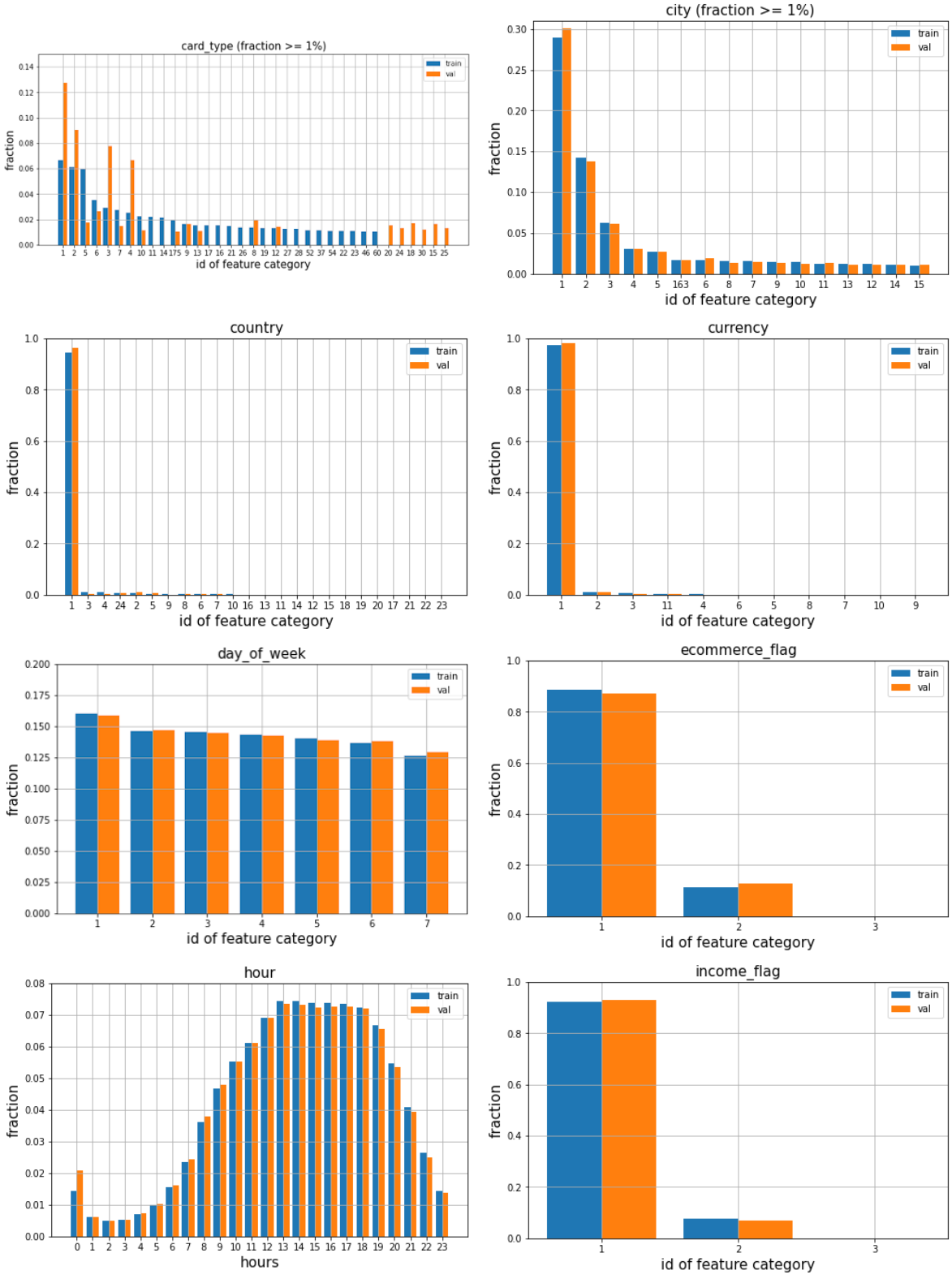
For all experiments, we use the AlfaBattle dataset. AlfaBattle is a dataset that was proposed by the organizers of the AlfaBattle 2.0 competition. It contains 400 million transactions made by one and a half million unique users. The feature list with description is presented in Table 1. There are no missing values in the dataset. The target for this task is whether a client will default or not. The transactions cover the user's one-year history. We split this dataset into training and validation sets in an 80/20 ratio. In all following tables, we report metrics on validation set.

Feature name	**Description**	**Cardinality**
currency	Transaction currency identifier	11
operation_kind	Transaction type identifier	7
card_type	Unique card type identifier	175
operation_type	Transaction type identifier	22
operation_type_group	Card transaction group identifier	4
ecommerce_flag	A sign of e-commerce	3
payment_system	Payment system type identifier	7
income_flag	Indication of debit/ deposit of funds	3
mcc	Unique point of sale type identifier	108
country	Transaction country identifier	24
city	Transaction city identifier	163
mcc_category	Transaction category identifier	28
day_of_week	Day of the week the transaction was made	7
hour	The hour when the transaction was made	24
weekofyear	Week when a transaction took place	53
days_before	Number of days before the loan	23
hour_diff	Number of hours since the last transaction	10
amnt	Normalised transaction amount	inf

Таблица 1: Description of transaction features of AlfaBattle dataset

4.2 Feature analysis

In Figure 2, one can see the distributions of these features.



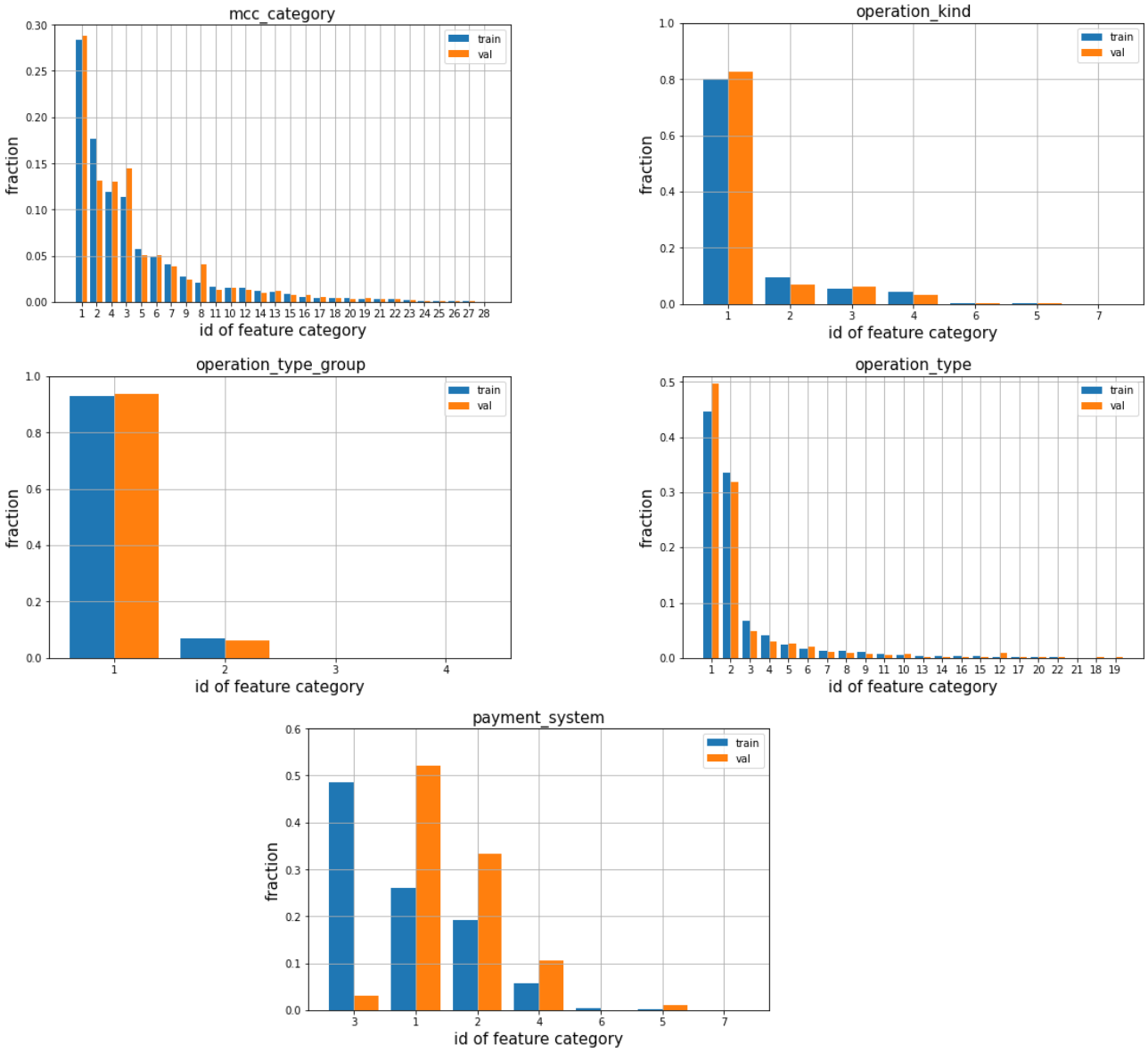


Рис. 2: Visualization of feature distributions of data on train and validation splits

4.3 Foundation Model

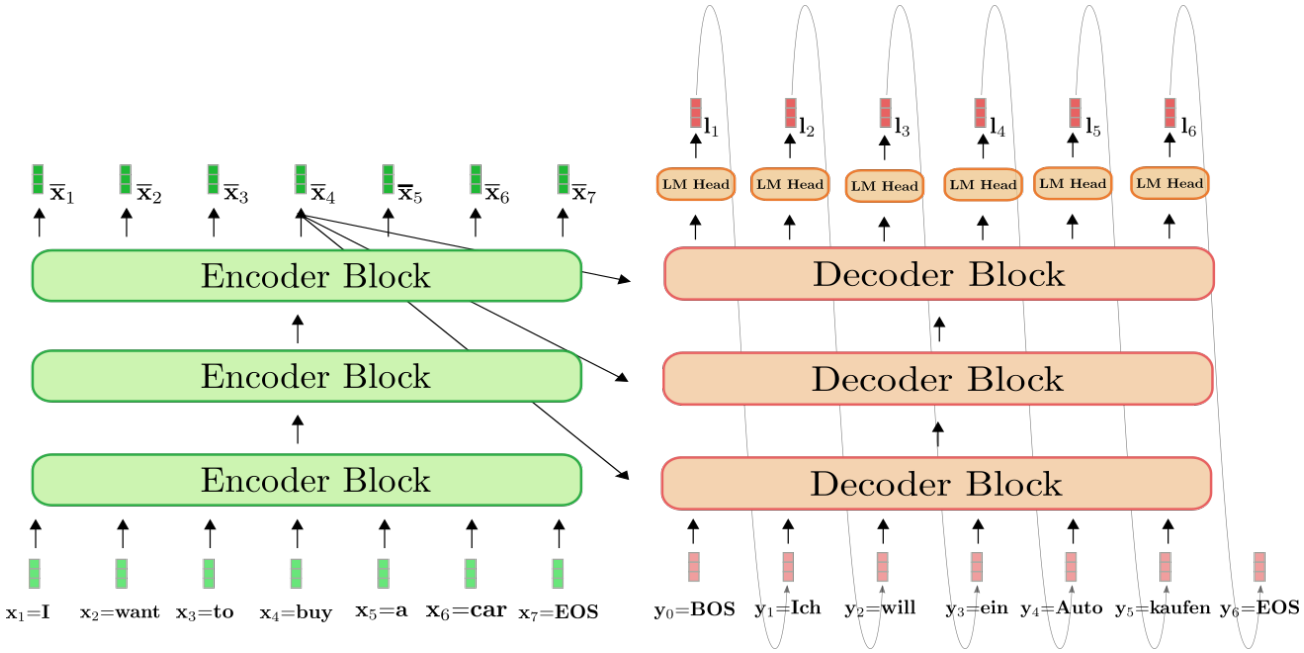


Рис. 3: Architecture of T5 model - transformer encoder-decoder architecture.

There are two types of models that can be candidates for the foundation language model: Transformer encoder-decoder models such as T5, T0, T0pp, Flan-T5, and Transformer decoder models such as GPT, GPT-Neo, and GPT-J. We have decided to choose encoder-decoder models, specifically FLAN-T5 models [36], because our early research experiments showed that the decoder's performance was poor compared to encoder-decoder models. Encoder-decoder models have the benefit of a context model as they have an encoder, and a generative model as the output is generated by the decoder. For the architecture, see Figure 3. We specifically chose the FLAN-T5 modification of T5 because it was pretrained on the maximum number of tasks from open-source models, and we believe that pretraining on a large number of tasks can benefit our idea.

There are three versions of the model that we utilize in our experiments: small, base, and large. The small version has 60 million parameters, the base version has 220 million parameters, and the large version has 770 million parameters.

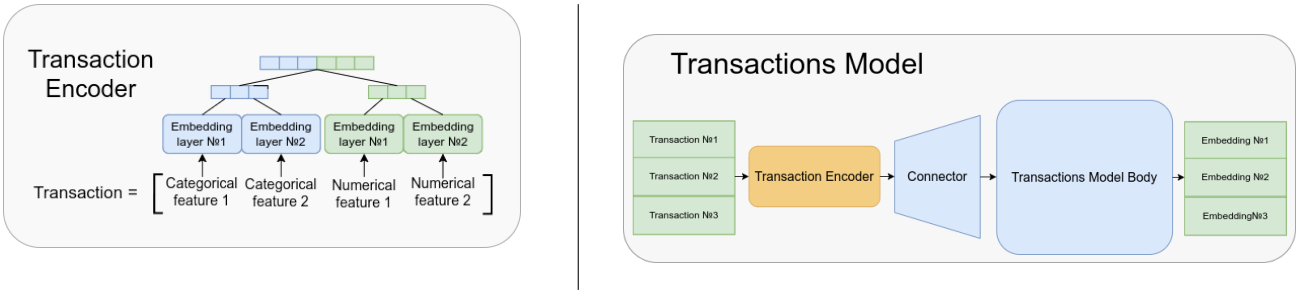


Рис. 4: Visualization of encoding procedure for one transaction and procedure on how all transactions are processed

4.4 Representation for event sequences

To determine the appropriate representation for event sequences, we conducted a series of experiments. The first series of experiments was devoted to checking the feasibility of using pretrained models for this task and selecting the model architecture. We conducted experiments on transformer encoders, transformer decoders, and transformer encoder-decoder architectures, which cover the main architectures applied to sequential data. The second series of experiments aimed to determine the most effective type of pretraining method.

4.4.1 Experiments with pretrained models

To address the need for multimodal capability in our transactional model, and due to the absence of pretrained event sequence models, we decided to select models that were pretrained on different modalities. We chose audio, image, and text models. Text models are relevant because our representation model will be further used as input for a language model. Audio models are relevant because audio itself can be considered an event sequence model. Image models offer a diverse range of architectures, and we can also utilize video models that can process images. For text architectures, we considered encoder models (BERT base, BERT large), encoder-decoder models (T5-small, T5-base), and decoder models (GPT base, GPT medium, GPT large). For image models, we considered ViT Base and ViT Large. Additionally, we selected video models: VideoMae Base and VideoMae Large. For audio modalities, we considered models such as whisper-tiny, whisper-base, whisper-medium, and speech-to-text transformer (S2T) small,

medium, and large. We trained the models in two regimes: scratch (training from random initialization) and finetune (fine-tuning pretrained models). Training the scratch model architecture allowed us to assess the architecture's performance, while finetuning helped us determine which modality's pretraining would benefit our model the most.

To validate the models, we assessed their performance on a default prediction task using the AUC score as the evaluation metric. We used the same training scheme for all experiments, employing Adam as the optimizer. For finetuning, we used a learning rate of 1e-5, and for training from scratch, we used a learning rate of 1e-4. We applied a linear warm-up of the learning rate for the first epoch, followed by linear decay to zero after 10 epochs. To ensure compatibility between the dimension of transaction embeddings and the dimension of pretrained model embeddings, we used a linear layer for text and audio modalities. Since image models cannot process sequences or mask padding layers, we applied a single layer of the Perceiver model (cross-attention to a fixed number of latent tokens) to address this issue. The results for text models can be found in Table 2, the results for image models are presented in Table 3, and the results for audio models can be seen in Table 4.

Model	**Number of parameters**	**Scratch**	**Finetune**
GPT2 Base	124M	0.7869	0.7745
GPT2 Medium	355M	0.7833	0.7798
GPT2 Large	774M	0.7747	0.7855
BERT Base	110M	Disconverged	0.769
BERT Large	335M	Disconverged	0.7652
T5 Small	60M	0.7721	0.7673
T5 Base	223M	0.7756	0.7731

Таблица 2: Results for solving default prediction task. Disconverged means the model converged to random prediction

Model	Number of parameters	Scratch	Finetune
ViT Base	85M	0.7822	0.7848
ViT Large	302M	0.7639	0.7791
VideoMAE Base	85M	0.7792	0.7725
VideoMAE Large	302M	0.7613	0.781

Таблица 3: Results for solving default prediction task using image models.

Model	Number of parameters	Scratch	Finetune
Whisper-tiny	29M	0.7892	0.7783
Whisper-small	153M	0.7894	0.7786
Whisper-medium	456M	0.7715	0.7815
S2T-small	12M	0.7894	0.7672
S2T-medium	30M	0.7854	0.7745
S2T-large	111M	0.7822	0.7848

Таблица 4: Results for solving default prediction task using audio models.

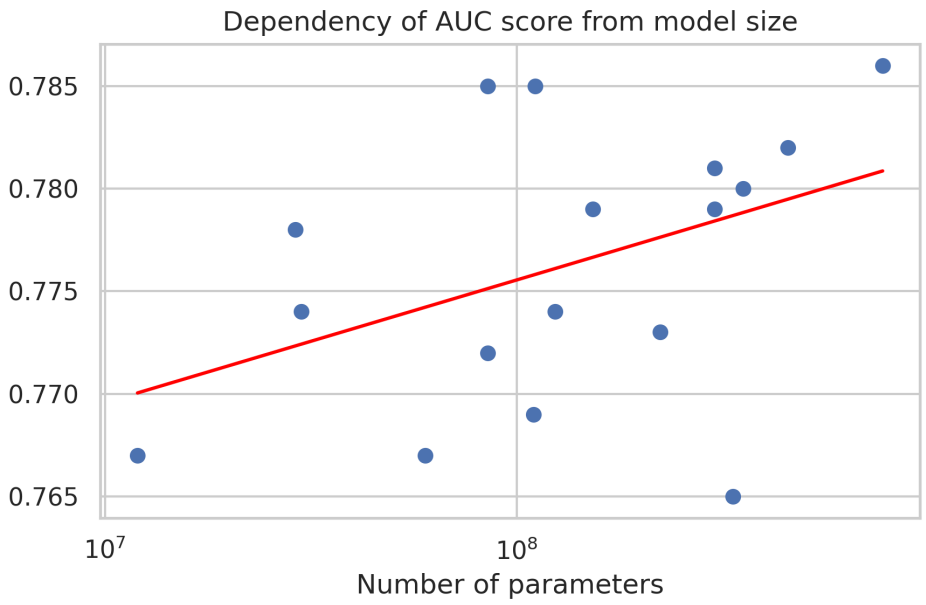


Рис. 5: Dependency of finetuning results on model size

Analysis. The results of the text models show that the decoder model performs better than the encoder models. The BERT model even diverged, and the T5

model shows worse results than the decoder model GPT2. Another indication that decoder models work better is the performance of the audio models. They are all decoder models, and their results are comparable to GPT2. The image models also demonstrate good performance, but we argue that it is due to the Perceiver model, which requires more parameters than a classical linear layer. Additionally, it can be observed that the results of finetuning improve as the number of parameters increases. This can be directly seen in Figure 5.

4.4.2 Monomodal pretraining

As we have chosen the transformer decoder architecture, it is important to determine which method of pretraining works best. Task-specific training can easily lead to overfitting on a specific task. Therefore, we decided to evaluate unsupervised and self-supervised approaches. By using unsupervised/self-supervised pretraining methods, we can create more robust representations that are capable of solving multiple tasks simultaneously. Hence, we compared two types of pretraining: Replace Token Detection and Contrastive Learning for Event Sequences (CoLES). We also compared transformer architecture with recurrent neural networks. For the pretraining method parameters, we used the settings from the CoLES paper.

To evaluate the pretrained representations, we selected three different settings: single-task training, multi-task training with next hour prediction and next amount prediction, and multi-task training with next hour prediction, next amount prediction, and next MCC prediction. For comparison, we also used four different fine-tuning regimes: fine-tuning all parameters, fine-tuning embedding parameters, fine-tuning encoder parameters, and fine-tuning only the head. As a baseline, we used the last event prediction, where the last known event is passed as the prediction for the next item. The results of these experiments are presented below.

Method / Finetuning	Scratch	All	Encoder	Embedding	Head
COLES-Whisper-small	0.721	0.718	0.712	**0.758**	**0.703**
COLES-GPT2-4	0.766	**0.779**	0.775	0.704	0.545
COLES-BERT-4	0.546	0.779	**0.775**	0.665	0.536
COLES-T5-8	0.702	0.735	0.744	0.699	0.564
COLES-GRU-1	**0.785**	0.776	0.771	**0.737**	0.685
COLES-LSTM-1	0.771	0.761	0.753	0.675	0.575
COLES-LSTM-4	0.766	0.757	0.753	0.691	0.564
RTD-GRU-1	**0.785**	0.773	0.768	0.655	0.579
CPC-GRU-1	**0.785**	0.774	0.769	0.713	0.636

Таблица 5: Results of finetuning on default prediction tasks

Single task training

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.078*	*0.078*	*0.078*	*0.078*
COLES-Whisper-small	**0.065**	0.0656	0.068	0.072
COLES-GPT2-4	0.066	0.066	0.069	0.077
COLES-GRU-1	0.066	0.066	**0.068**	0.074
COLES-LSTM-1	0.066	0.066	0.068	0.073
COLES-LSTM-4	0.066	**0.0663**	0.072	0.076
RTD-GRU-1	0.066	0.067	0.068	0.074
CPC-GRU-1	0.066	0.066	0.068	**0.073**

Таблица 6: Results of finetuning on next amount prediction task

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.407*	*0.407*	*0.407*	*0.407*
COLES-Whisper-small	**0.489**	**0.488**	0.465	0.451
COLES-GPT2-4	0.485	0.483	0.461	0.426
COLES-GRU-1	0.480	0.479	0.468	0.439
COLES-LSTM-1	0.481	0.480	**0.469**	**0.451**
COLES-LSTM-4	0.484	0.484	0.445	0.396
RTD-GRU-1	0.480	0.479	0.467	0.440
CPC-GRU-1	0.480	0.479	0.469	0.446

Таблица 7: Results of finetuning on next MCC prediction task

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.531*	*0.531*	*0.531*	*0.531*
COLES-Whisper-small	**0.641**	0.637	0.422	0.466
COLES-GPT2-4	0.257	0.255	0.095	0.333
COLES-GRU-1	0.332	0.496	0.382	0.367
COLES-LSTM-1	0.468	0.435	0.306	0.357
COLES-LSTM-4	0.454	**0.653**	0.397	0.367
RTD-GRU-1	0.508	0.410	0.492	0.400
CPC-GRU-1	0.332	0.501	0.462	0.337

Таблица 8: Results of finetuning on next hour prediction task

Multi-task training: next hour prediction and next amount prediction

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.0788*	*0.0788*	*0.0788*	*0.0788*
COLES-Whisper-small	***0.0664***	**0.0665**	**0.0692**	**0.0725**
COLES-GPT2-4	*0.0679*	0.0697	0.0722	0.0776
COLES-GRU-1	*0.0676*	0.0690	0.0716	0.0740
COLES-LSTM-1	***0.0678***	0.0695	0.0711	0.0734
COLES-LSTM-4	*0.0675*	0.0685	0.0733	0.0764
RTD-GRU-1	*0.0675*	0.0687	0.0715	0.0746
CPC-GRU-1	0.0676	0.0683	0.0706	0.0733
Best scratch (GPT-2)	0.0627	0.0627	0.0627	0.0627

Таблица 9: Results of finetuning on next amount prediction task

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.4075*	*0.4075*	*0.4075*	*0.4075*
COLES-Whisper-small	***0.4826***	**0.4829**	0.4635	0.4517
COLES-GPT2-4	0.4747	0.4722	0.4557	0.4233
COLES-GRU-1	0.4746	0.4728	0.4616	0.4392
COLES-LSTM-1	0.475	0.4725	0.4644	0.4519
COLES-LSTM-4	0.4779	0.4757	0.4374	0.3969
RTD-GRU-1	0.4748	0.4725	0.4647	0.4406
CPC-GRU-1	0.4744	0.4725	**0.4656**	0.4463
Best scratch (GPT-2)	0.4883	0.4883	0.4883	0.4883

Таблица 10: Results of finetuning on next MCC prediction task

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.5319*	*0.5319*	*0.5319*	*0.5319*
COLES-Whisper-small	0.5497	0.5565	**0.5643**	0.5305
COLES-GPT2-4	0.5473	0.5565	0.5169	0.5891
COLES-GRU-1	0.5473	0.5541	0.4992	0.4178
COLES-LSTM-1	0.5497	0.5565	0.4897	0.4362
COLES-LSTM-4	0.5652	0.5565	0.4383	0.36
RTD-GRU-1	0.545	0.5557	0.4883	0.3808
CPC-GRU-1	0.5536	**0.5576**	0.4631	0.3781
Best scratch (GPT-2)	0.6162	0.6162	0.6162	0.6162

Таблица 11: Results of finetuning on next hour prediction task

Multi-task training: next hour prediction, next amount prediction and next MCC prediction

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.0788*	*0.0788*	*0.0788*	*0.0788*
COLES-Whisper-small	**0.0667**	**0.0666**	**0.0692**	**0.0725**
COLES-GPT2-4	0.0684	0.0712	0.0686	0.0744
COLES-GRU-1	0.0687	0.0699	0.0690	0.0735
COLES-LSTM-1	0.0685	0.0704	0.0688	0.0765
COLES-LSTM-4	0.0677	0.0748	0.0683	0.0748
RTD-GRU-1	**0.0683**	0.0704	0.0691	0.0734
CPC-GRU-1	0.0684	0.0700	0.0687	0.0778
Best scratch (GPT-2)	0.0627	0.0627	0.0627	0.0627

Таблица 12: Results of finetuning on next amount prediction task

	All	Encoder	Embedding	Head
Baseline	*0.4075*	*0.4075*	*0.4075*	*0.4075*
COLES-Whisper-small	**0.4857**	**0.485**	0.4624	0.4517
COLES-GPT2-4	0.4779	0.4503	0.4775	0.4392
COLES-GRU-1	0.4746	0.4629	0.4726	**0.4528**
COLES-LSTM-1	0.4754	0.4619	0.4724	0.4023
COLES-LSTM-4	0.4794	0.4231	**0.4789**	0.4403
RTD-GRU-1	0.4747	0.4635	0.7401	0.4464
CPC-GRU-1	0.4741	0.4642	0.4725	0.4259
Best scratch (GPT-2)	0.4883	0.4883	0.4883	0.4883

Таблица 13: Results of finetuning on next MCC prediction task

Method / Finetuning	All	Encoder	Embedding	Head
Baseline	*0.5319*	*0.5319 0.5319*	*0.5319*	*0.5319*
COLES-Whisper-small	**0.6125**	0.6173	0.5575	0.5311
COLES-GPT2-4	**0.6305**	**0.6324**	**0.6313**	0.5045
COLES-GRU-1	0.6047	0.5419	0.62	0.4769
COLES-LSTM-1	0.6033	0.5174	0.6144	-
COLES-LSTM-4	0.622	0.3107	0.6176	0.5328
RTD-GRU-1	0.5967	0.549	0.6089	0.4868
CPC-GRU-1	0.6033	0.5276	0.6122	**0.611**
Best scratch (GPT-2)	0.6162	0.6162	0.6162	0.6162

Таблица 14: Results of finetuning on next hour prediction task

Analysis Results of experiments showing that there is no significant difference between different methods of pretraining in terms of metrics. Also, we see that encoder-decoder and encoder models perform worse than these architectures. This confirms yet again the results of the pretrained model experiment. As results are showing that the results of pretraining pretty similar and decoder architecture dominates other architecture we decided to stop our choice on transactional model: whisper-tiny pretrained on next item prediction task and use it for text interfaces.

4.5 Text Interfaces

To explore the feasibility of constructing text interfaces, we conducted a series of experiments to assess the ability of text interfaces to solve extractive and predictive tasks, as well as their potential for multitasking.

In all subsequent experiments, we used the following training settings. We employed the Adam optimizer with a learning rate of 1e-4. A linear warm-up of the learning rate was conducted for 100 steps, followed by linear decay until the end of training, which lasted for 15 epochs. The batch size was set to 32.

4.5.1 Extractive Questions

To evaluate extractive questions, we focused on tasks related to the prediction of merchant category codes (MCC). Specifically, we aimed to predict

the most frequent MCC value for a given user. For binary questions, we asked the model whether a particular MCC code is the most popular for the user. For multiple-choice and open-ended questions, we asked the model to identify the merchant category code that is most popular for the user. The reported metric is weighted accuracy. "N/A"indicates that the model performs at the same level as random prediction. The results for binary questions are presented in Table 15, the results for multiple-choice questions are presented in Table 16, and the results for open-ended questions are presented in Table 17.

	All		**LM**		**Connector**	
Metrics	accuracy	f1-score	accuracy	f1-score	accuracy	f1-score
flan-t5-small	**0.964**	0.963	0.946	0.945	N/A	N/A
flan-t5-base	0.948	0.947	0.956	0.955	N/A	N/A

Таблица 15: Results for context binary questions for MCC feature

	All		**LM**		**Connector**	
Metrics	*accuracy*	*f1-score*	*accuracy*	*f1-score*	*accuracy*	*f1-score*
flan-t5-small	0.941	0.940	0.932	0.93	N/A	N/A
flan-t5-base	**0.961**	0.957	0.953	0.947	N/A	N/A

Таблица 16: Results for context multichoice questions for MCC feature.

	All		**LM**		**Connector**	
Metrics	*accuracy*	*f1-score*	*accuracy*	*f1-score*	*accuracy*	*f1-score*
flan-t5-small	0.771	0.727	0.685	0.634	N/A	N/A
flan-t5-base	0.736	0.682	**0.764**	0.721	N/A	N/A

Таблица 17: Results for context open-ended questions for MCC feature.

Analysis From the results, it can be observed that the performance for binary and multiple-choice settings is nearly perfect. However, the performance for open-ended answers is not as good, with results close to 0.7, which is relatively low. Nevertheless, there is an improvement as the model size increases, indicating that increasing the size of the model can be beneficial. It is also noteworthy

that finetuning only the connector layer is not sufficient. We speculate that the transactional modality and text are too different to be disentangled using just one layer.

4.5.2 Predictive Questions

To evaluate predictive questions, we also considered the prediction of merchant category codes (MCC) and, additionally, the prediction of the number of transactions in the next 30 days. For binary questions, we asked the model whether the next transaction would be the most popular one. For multiple-choice and open-ended questions, we asked the model to predict the merchant category code for the next transaction. In the case of predicting the number of transactions in the next 30 days, we asked the model whether the number of transactions would be more than the median in the binary setting. In the multi-task setting, we asked the model about the number of transactions in the next 30 days. We also included training baselines in the classical format to compare the results of the text interface against the classical model. As the classical model, we selected a transaction model with an additional head that solves the specific task. The reported metric is accuracy. The results for binary questions are presented in Table 15, the results for multiple-choice questions are presented in Table 16, and the results for open-ended questions are presented in Table 17.

	All		**LM**		**Transaction model**	
Model / Task	MCC	30 days	MCC	30 days	MCC	30 days
classification model	*0.7885*	*0.8512*	*0.7885*	*0.8512*	*0.7885*	*0.8512*
flan-t5-small	**0.7897**	**0.8539**	0.7853	0.8434	0.7803	0.8515
flan-t5-base	0.7886	0.8539	0.7858	0.8429	0.7803	0.8515

Таблица 18: Results for predictive binary questions for MCC feature

	All		LM		Transaction model	
Model / Task	MCC	30 days	MCC	30 days	MCC	30 days
classification model	*0.8299*	*0.939*	*0.8299*	*0.939*	*0.8299*	*0.939*
flan-t5-small	**0.828**	**0.935**	0.825	0.930	0.780	0.610
flan-t5-base	0.827	0.932	0.811	0.926	0.780	0.610

Таблица 19: Results for predictive multichoice questions for MCC feature.

	All		LM		Transaction model	
Model / Task	MCC	30 days	MCC	30 days	MCC	30 days
classification model	*0.573*	0.444	*0.573*	*0.444*	*0.573*	*0.444*
flan-t5-small	**0.607**	0.375	0.603	0.3428	0.595	0.175
flan-t5-base	0.601	0.385	0.571	**0.397**	0.595	0.175

Таблица 20: Results for predictive open-ended questions for MCC feature.

Analysis From the results, one can observe that our model performs well, even outperforming the classical format models. Additionally, it can be seen that when the output is multiple-choice or open-ended, training only the transaction model results in worse performance. We argue that this is because in the binary case, where the number of possible answers is very low, the model can learn the correct answers without finetuning. However, when the number of unique answers is large, finetuning the language model becomes necessary.

4.6 Multi-Task Training

We conducted the following experiments to investigate whether it is possible to achieve the claimed capability of the model to solve multiple tasks simultaneously. We trained the model to solve several task simultaneously. For each batch we sample one of several tasks and ask the model to solve the problem.

4.6.1 Extractive Tasks

For the multi-task setting, we selected several tasks, which are most hard to solve by results of our experiments: amount prediction, MCC prediction, and

MCC category prediction. The results are presented in Table 21.

Модель	Amount	MCC code	MCC category
flan-t5-small	*random guess*	*random guess*	*random guess*
flan-t5-base	**0.019**	0.697	0.744
flan-t5-large	**0.022**	0.637	0.684

Таблица 21: Results for training text interfaces in multi-task setting for extractive tasks

Analysis It can be observed that the results for the smallest model are not better than random predictions. However, as the model size increases, the results improve and become quite close to the results obtained in the single task setting.

4.6.2 Predictive Tasks

For the predictive tasks, we considered four different tasks: default prediction, next hour prediction, next amount prediction, and next MCC prediction. We trained the model independently for each task and also using pairs of tasks. The results are presented in Table 22.

	Default	Next MCC	Next Amount	Next Hour
Single task	**0.783**	0.756	**0.703**	**0.7043**
MCC + Amount	-	0.750	0.6549	-
MCC + default	0.7715	0.7554	-	-
4 tasks	0.7616	**0.7585**	0.6893	0.651

Таблица 22: Results for training text interfaces in multi-task setting for predictive tasks

Analysis It can be observed that the model is able to solve several tasks simultaneously. Additionally, it should be noticed that the results decrease as the number of tasks decreases. However, this is a common issue for language models and can be solved by increasing the number of parameters.

4.7 Quality of LLM on NLG Tasks

To investigate whether finetuning harms the performance of the language model on natural language understanding tasks, we conducted the following experiment. We selected 12 tasks from the BigBench dataset. The Beyond the Imitation Game Benchmark (BIG-bench) is a collaborative benchmark intended to assess large language models and predict their future capabilities. From our results, we observed that the quality of the finetuned model does not change significantly compared to the non-finetuned model for most tasks. However, interestingly, the quality even improved for the Vitamin C fact verification task. To understand how this improvement occurred, we examined the task in more detail.

Vitamin C aims to measure the model's ability to determine the truthfulness of a given claim based on related external evidence. To succeed, the model must rely on the information provided in the evidence, even if it contradicts prior knowledge that it acquired during training.

Source: Based only on the information contained in a brief quote from Wikipedia, determine whether the related claim is True, False, or Neither. Use Neither when the Wikipedia quote does not provide sufficient information to resolve the question.
Passage: Mercedes-Benz: Mercedes-Benz is a global aircraft manufacturer and a division of a British airline. Claim: Mercedes-Benz is a division of an airline. True, False, or Neither? OPTIONS: - True; - False; - Neither
Target: True
Predicted: True

Analysis It can be seen that the format of the answer is the same as in the binary setting. Therefore, by finetuning on binary tasks, the model learned the concepts of True and False a little better.

5 Discussion and conclusion

In this work, we have developed text interfaces for sequential data. We conducted extensive research on the construction of text interfaces and identified that decoder architectures work best for this purpose. We propose a scheme for constructing text interfaces on event sequences and have found that the quality of the results is comparable to that of classical models.

On the global research landscape, our work is the first to propose the application of text interfaces for event sequences of data.

One limitation of this research is that it may not be readily accessible to ordinary users. The quality of the answers improves with the size of the model, and only a small number of people have the necessary resources for storing and training such models. Additionally, the absence of a large open-source dataset makes it challenging to conduct more extensive research.

Acknowledgements

- Ivan Oseledets and Vadim Strijov for supervision
- Elizaveta Goncharova for reviewing my work
- Maxim Zubkov for providing guidance on NLP world and fruitful discussions.

Список литературы

[1] A. Yeshchenko and J. Mendling, "A survey of approaches for event sequence analysis and visualization using the esevis framework," *arXiv preprint arXiv:2202.07941*, 2022.

[2] D. Babaev, N. Ovsov, I. Kireev, M. Ivanova, G. Gusev, I. Nazarov, and A. Tuzhilin, "Coles: Contrastive learning for event sequences with self-supervision," in *Proceedings of the 2022 International Conference on Management of Data*, pp. 1190–1199, 2022.

[3] T. M. Mitchell *et al.*, *Machine learning*, vol. 1. McGraw-hill New York, 2007.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[5] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[6] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," *Advances in Neural Information Processing Systems*, vol. 35, pp. 36479–36494, 2022.

[7] R. Gandikota, J. Materzynska, J. Fiotto-Kaufman, and D. Bau, "Erasing concepts from diffusion models," *arXiv preprint arXiv:2303.07345*, 2023.

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

[9] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[10] I. Rubachev, A. Alekberov, Y. Gorishniy, and A. Babenko, "Revisiting pretraining objectives for tabular deep learning," *arXiv preprint arXiv:2207.03208*, 2022.

[11] Y. Gorishniy, I. Rubachev, and A. Babenko, "On embeddings for numerical features in tabular deep learning," *arXiv preprint arXiv:2203.05556*, 2022.

[12] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, "Tabtransformer: Tabular data modeling using contextual embeddings," *arXiv preprint arXiv:2012.06678*, 2020.

[13] R. Cholakov and T. Kolev, "The gatedtabtransformer. an enhanced deep learning architecture for tabular modeling," *arXiv preprint arXiv:2201.00199*, 2022.

[14] I. Padhi, Y. Schiff, I. Melnyk, M. Rigotti, Y. Mroueh, P. Dognin, J. Ross, R. Nair, and E. Altman, "Tabular transformers for modeling multivariate time series," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3565–3569, IEEE, 2021.

[15] D. Babaev, M. Savchenko, A. Tuzhilin, and D. Umerenkov, "Et-rnn: Applying deep learning to credit loan applications," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2183–2190, 2019.

[16] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 95–104, 2018.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez,

Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[18] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18932–18943, 2021.

[19] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[20] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International Conference on Machine Learning*, pp. 5156–5165, PMLR, 2020.

[21] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 11106–11115, 2021.

[22] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *International Conference on Machine Learning*, pp. 27268–27286, PMLR, 2022.

[23] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[24] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[25] Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, J. Wei, X. Wang, H. W. Chung, D. Bahri, T. Schuster, H. S. Zheng, *et al.*, "Ul2: Unifying language learning paradigms," 2022.

[26] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[27] M. Sharma, A. Gogineni, and N. Ramakrishnan, "Innovations in neural data-to-text generation," *arXiv preprint arXiv:2207.12571*, 2022.

[28] A. P. Parikh, X. Wang, S. Gehrmann, M. Faruqui, B. Dhingra, D. Yang, and D. Das, "Totto: A controlled table-to-text generation dataset," *arXiv preprint arXiv:2004.14373*, 2020.

[29] D. Bertsimas, K. V. Carballo, Y. Ma, L. Na, L. Boussioux, C. Zeng, L. R. Soenksen, and I. Fuentes, "Tabtext: a systematic approach to aggregate knowledge across tabular data structures," *arXiv preprint arXiv:2206.10381*, 2022.

[30] V. Borisov, K. Seßler, T. Leemann, M. Pawelczyk, and G. Kasneci, "Language models are realistic tabular data generators," *arXiv preprint arXiv:2210.06280*, 2022.

[31] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer, "Tabbie: Pretrained representations of tabular data," *arXiv preprint arXiv:2105.02584*, 2021.

[32] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," *arXiv preprint arXiv:2005.08314*, 2020.

[33] A. Drozdov, N. Schärli, E. Akyürek, N. Scales, X. Song, X. Chen, O. Bousquet, and D. Zhou, "Compositional semantic parsing with large language models," *arXiv preprint arXiv:2209.15003*, 2022.

[34] S. Duong, A. Lumbreras, M. Gartrell, and P. Gallinari, "Learning from multiple sources for data-to-text and text-to-data," *arXiv preprint arXiv:2302.11269*, 2023.

[35] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[36] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[37] Y. Hao, H. Song, L. Dong, S. Huang, Z. Chi, W. Wang, S. Ma, and F. Wei, "Language models are general-purpose interfaces," *arXiv preprint arXiv:2206.06336*, 2022.

[38] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, *et al.*, "Socratic models: Composing zero-shot multimodal reasoning with language," *arXiv preprint arXiv:2204.00598*, 2022.

[39] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, *et al.*, "Flamingo: a visual language model for few-shot learning," *arXiv preprint arXiv:2204.14198*, 2022.

[40] D. Surís, S. Menon, and C. Vondrick, "Vipergpt: Visual inference via python execution for reasoning," *arXiv preprint arXiv:2303.08128*, 2023.

[41] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *arXiv preprint arXiv:2302.04761*, 2023.

[42] S. Geng, S. Liu, Z. Fu, Y. Ge, and Y. Zhang, "Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5)," in *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 299–315, 2022.