Telemetry Encoder with Language Models

Francisco Caldas
francisco.caldas@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

July 2022

Abstract

Everyday, there are huge amounts of system logs collected by software applications, which record detailed information of computational events that can be used for monitoring, administering, troubleshooting or keeping records of the system. Outsystems' software, called Service Studio, is no exception. However, the nature of these Service Studio logs is such that it is hard to extract meaningful insights from these data. The data is structured, has a lot of noise, and represents very low-level events which is hard to abstract and to gather more meaningful insights to the user-level. In this work, we propose a BERT-like model to learn to represent this data as embeddings in a self-supervised way, in order to later serve different machine learning models. We first pretrain our model using a custom made task, and interpret its predictions using a gradient saliency visualization. We also fine tune it for a supervised dataset called Atomic Tasks, and show the benefit of our pretraining step. Finally we compare the learned embeddings with the different UI areas of the software and see that some of these areas were encoded in the pretrained embeddings. We conclude that the BERT architecture is beneficial to learn useful representations of this type of data that can serve different fine-tuning tasks.
Keywords: Outsystems, BERT, Log Events, Embeddings, NLP

1. Motivation

Companies like Outsystems (OS) created low-code development platforms as an answer to accelerate the process of creating working applications and the training of new developers. Outsystems' platform does not rely on a single programming language directly, but on a User Interface (UI) that allows developers to build all the components of a modern application such as UI, business processes, logic, data models, etc. The Artificial Intelligence team at OS aims to guide users through their development process using the expertise learned from millions of anonymized code patterns to suggest the right tools and patterns for each development situation. Therefore, it is crucial to learn good representations of these interactions to better learn the code patterns that will later benefit users in their coding experience.

For each user interaction, Outsystems' platform, named Service Studio, generates a group of logs that describe the interaction and the background system events that are automatically generated (see an example of a single event in Figure 1). Each event is a tabular semi-structured datapoint that holds the event's information regarding its type, its variables and other useful information.

The motivation of the thesis is to create a numerical representation of these events that encodes useful information to serve as input to train future mathematical models (statistical models, deep learning models, etc.), starting from a popular idea in the area of anomaly detection: to treat these events as text and to apply natural language processing techinques to extract useful information from them.

Starting from this idea of applying state of the art NLP techniques to learn embeddings (or representations) of these events, although they cannot be considered natural language, the objective is to train language models using these logs and then extract representations from them, arguing in the end that these representations will be useful for tasks of interest to Outsystems and different from those in which the language models were trained.

2. Topic Overview

Our problem consists in learning embeddings to represent datapoints of the already mentioned type represented in Figure 1, where the meaning and type of some fields is also explained. However, it is necessary to introduce the reader to the important fields present in the dataset such as INSTALLER_ID and SESSION_ID, which identify each event together with its timestamp named EVENT_TIMESTAMP; and EVENT_TYPE, EVENT_NAME, DOMAIN_DATA

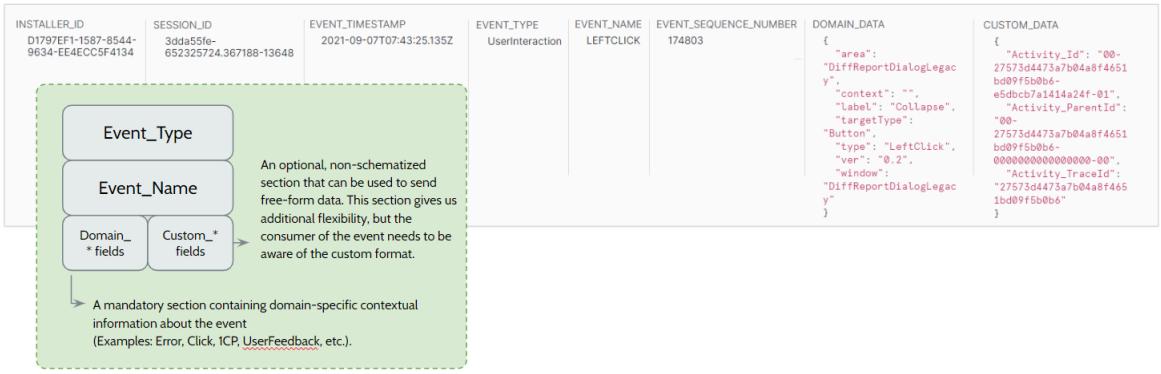

Figure 1: An example of one event generated by Service Studio

and CUSTOM_DATA that describe the properties of the event. This type of data is in a gray area comparing to some popular fields in ML, which makes it hard to categorize our problem into a common ML topic. It could argued that the data should be classified as a tabular, a semi-structured, or a system log dataset.

First, the data is tabular, since it has some categorical features like EVENT_TYPE, but it also has some semi-structured fields, and categorical fields with high cardinality. Second, it could be categorized as a semi-structured dataset since it has JSON fields like DOMAIN_DATA or CUSTOM_DATA, but this category would not be very suitable for a sequential dataset (with a considerable amount of important user behavior implicit in the temporal order of the events) that is generated in a log-like fashion, with a big vocabulary of possible words, most of them from a programming domain. It could also be regarded as a common system log dataset since it has variables, timestamps and system generated events. However, usually the format of logs is quite close to a sentence of Natural Language (e.g. "variable x was not updated.") and our data format is clearly not close to Natural Language since it is mostly structured and it is almost only composed of nouns and verbs (and no discernible semantic grammar or order).

This heterogeneity of our problem makes deep learning a good hypothesis to tackle it, since deep learning has been shown to be able to effectively learn datasets of almost any type. More recently, the Transformer architecture has even shown some good results in learning multi-modal problems, which also justifies its choice as the best contender to approach our problem.

In recent years, some major breakthroughs in learning algorithms and hardware, as well as an increase in the amounts of data available, have sparked an increasing interest in artificial intelligence. These breakthroughs in hardware, Machine Learning algorithms and large datasets have facilitated what some scholars call the Artificial Intelligence Revolution [12].

A Transformer [17] is a deep learning model originally proposed as a sequence-to-sequence model [16] for machine translation, but its success in NLP tasks sparked great interest in different fields. In Computer Vision, Transformers have been used, for example, for image classification [4, 1], object detection [18, 1] and image generation [7]. In Audio Applications, speech recognition [3, 2], speech synthesis [10, 22] and music generation [20] have also been successful. And many other fields including Multimodal Applications such as visual question and answering (Q&A) and commonsense reasoning [9], caption generation [15], speech-to-text translation [5] and text-to-image generation [13].

Transformers were inspired on recurrent neural network models (such as LSTMs), but without needing to process the input data sequentially. It contains multiple self-attention layers, and can be conventionally learned with gradient decent.

Transformers were such a success in text, that it was not long until the architecture was applied to images, which also showed great success. These influences between domains is very common in the field of ML. In the case of NLP techniques, it is also common for successful approaches being then applied to sequential data that is not textual. For similar use cases to ours, either in tabular, structured or log data, Transformers have been shown to be competitive models to tackle these types of problems .

In conclusion, our problem is in an intersection of different topics in ML, which motivates the use of Transformers because of it flexibility with different data types as well as state of the art performance in a big ranage of tasks.

3. Objectives

The goal of the thesis is to develop a Transformer model using the logs from Outsystems' Service Studio. From the representations learned during the pre-training of the model, a meaningful fine-

tuning task will be designed to test the usefulness of the embeddings. After obtaining the embeddings (using different strategies) we will discuss their quality and the potential use in future research.

4. Outline

This work comprises of a total of eight sections. Section 5 presents the methodology and the reasoning behind the main choices we made regarding the training steps. Section 6 exposes and discusses the obtained results and the additional experiments that sprung from the initial results. Finally, Section 7 wraps up this work by taking a final overview and suggesting possible future directions. The main contribution of this thesis is the introduction of a novel approach to encode the Telemetry data from a low-code software, more specificaly Outsystems' Service Studio.

5. Methodology

Before answering our research questions, we must first outline our methodology. Our decisions were heavily inspired by the related work on the applications of transformers in different domains. However, because of time constraints of the project (specially because of big training times of deep learning models), finding the best possible solution must be sacrificed to finding a working solution at all. Therefore, we relied mainly on a depth-first search approach, which was fundamental to the success of this work and will be key to understand the chosen methodology.

5.1. Hugging Face

Machine Learning research, specially deep learning, requires streamlined workflows to be done efficiently. Without the help of optimized libraries which help on computation efficiency, algorithm implementations and debug/ monitoring, the process of training a deep learning model can be very error prone, slow and hard to replicate. The complexity of a problem of this nature, from data collection, data cleaning, data preprocessing, and training to evaluating, monitoring, loading and saving increases even when adapting to different tasks, models and datasets.

Hugging Face's library *Transformers* [19] aims to solve all of this problems in an integrated python library. As the authors state:

> "Transformers is an open-source library with the goal of opening up these [research in transformers] advances to the wider machine learning community. The library consists of carefully engineered state-of-the art Transformer architectures under a unified API. Backing this library is a curated collection of pretrained models made by and available for the community. Transformers is designed to be extensible by researchers, simple for practitioners, and fast and robust in industrial deployments. The library is available at https://github.com/huggingface/transformers."

All the models, pretrained or randomly initialized, we will train will be provided by this library. Every model in the library is fully defined by a tokenizer (transforms string inputs into a sequence of tokens from a vocabulary), a transformer (transforms a sequence of indices from a dictionary of tokens to contextual embeddings), and a head (uses contextual embeddings to make a task-specific prediction).

5.2. Architecture

From the huge library of models from Hugging Face, we chose 6 models (see Table 1) trained using the same pretraining MLM task strategy. Out of the 6 architectures, 2 are variations from the original BERT, 3 from RoBERTa (a variation on BERT by removing the NSP and training with much larger mini-batches and learning rates) and 1 from ALBERT (a Lite version of BERT with less parameters and faste training). Their Masked language modeling (MLM) are all based on BERT where, from the input, 15% of the tokens are selected, and out of these are: (1) replaced by the [MASK] token 80% of the time (2) switched for a random token 10% of the time (3) unchanged 10% of the time. Then, this modified input is used to predict the original selected tokens with cross entropy loss.

However, most downstream tasks will depend more on learned relations between full events, and not between individual tokens. This is important because changing 15% of the tokens of each event will probably force the model to rely on learning the structure of each individual event and not the relations between them. For example, the model will easily learn that for every `UserInteraction` where (e.g.) the token `User##` is masked but `##Interaction` isn't, the most likely word is `UserInteraction` , whithout having to learn the relations between the neighboring events, which are the most interesting to us. Therefore, we implemented a custom event masking called Full Event Masking (FEM), where we select 15% of the events that were concatenated in the input, and fully mask them (using the [MASK] token). Therefore, the model is forced to learn to predict all the tokens from the other events. A similar approach can be found in [21].

Finally, in order to rapidly choose our contender out of the 6 elected architectures, we pretrained all of the models for 1k steps of the pretraining MLM task, with a batch size of 8 to choose the best con-

Model Name	Citation
microsoft/deberta-v3-xsmall'	[6]
'microsoft/deberta-v3-small'	[6]
albert-base-v2'	[8]
'huggingface/CodeBERTa-small-v1'	[19]
'distilroberta-base'	[11]
distilbert-base-uncased'	[14]

Table 1: Referrences for the Huggingface model names used in this work.

tender for the next experiments.

We will show later the impact of this pretraining step on the model's ability to learn the fine tuning task.

5.3. Fine Tuning Tasks

For our fine tuning task, we will use the atomic tasks, since we have an unlimited amount of data, because it is a simple, rule based model. It also seems the most generic task a priori, which coincides with our final goal. To evaluate the pretrained embeddings, we will also rely on the productivity areas dataset to explore some properties that might be encoded in the emebddings. In this section we argue why we chose the Atomic Tasks dataset for the fine tuning task.

First, the goal of predicting Stuck Moments is probably the closest task to a real application of our work in Service Studio. However, it is hard to define and measure a true stuck moment of a developer. Besides the probable low signal-to-noise ratio of this option, it would also be very time consuming to build this dataset.

Second, the Productivity Areas dataset, despite being a good contender, does not have a lot of granularity. This low number of categories would probably simplify the signal too much and give rise to not very meaningful embeddings.

Lastly, Atomic Tasks, as the first level of abstraction above the raw telemetry events, still have a considerable number interpretable categories (43), which makes it the best contender. However, some of rules that define these tasks can span over hundreds of events, which probably would not fit into a single input of a common Transformer, but this is not the rule but the exception (on average, 512 tokens of input correspond to 20 consecutive events). Finally, as this dataset is rule-based, we can generate as many labels as we want.

So, for all these reasons, we chose this ground truth for fine tuning our model.

5.4. Evaluating and Differentiating

In this work we expect to evaluate different aspects of our solution.

First, we want to confirm that our approach of treating this data as textual is good enough to enable a proper learning by a transformer architecture, either during pretraining or fine tuning. We can look at the training and evaluation set losses during training for this).

Second, we also want to understand if the our pretraining task (either with the default or the FEM) helps the model learn downstream fine-tuning tasks faster. We can test this by fine tuning a model that we pretrained and a randomly initialized model and see if the loss is impacted by our pretraining.

Third, we want to check that our model is capable of learning to classify the atomic tasks of each event. We rely mainly on f1-score for this purpose.

Lastly, we want to see if useful information was encoded in the embeddings during pretraining. We will check this by first plotting the embeddings using the dimensionality reduction technique PCA; and then by seeing if the learned embeddings are correlated with a different dataset (productivity areas). We do this by training a Random Forest using the embeddings as the input and analyzing its performance. If the model is able to learn, it means that information regarding the main UI and UX areas was also encoded in the embeddings.

6. Experiments

6.1. Pretraining

As explained in Section 5.2 we pretrained all the contender models on the default MLM task described in Bert for 1k iterations. We ran each architecture two times with the corresponding original parameteres initialized and averaged its losses. The results can be seen in Figure 2

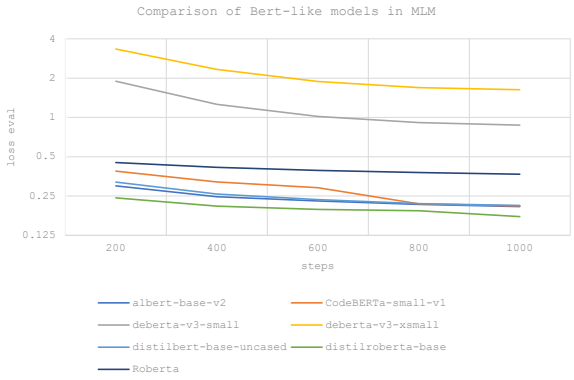

Figure 2: 1k iterations of each architecture

The architecture that showed best performance on the first steps was clearly *distilroberta-base*. This will be the architecture in which we will run all of our further experiments. More tests could be done to find a more fitting architecture, but we chose the architecture as fast as possible to give more time to all next steps.

After this, we pretrained the model for 20k itera-

tions as shown in Figure 3. The training and evaluation loss improve during the 20k iterations, thus showing no apparent problems of overfittng.

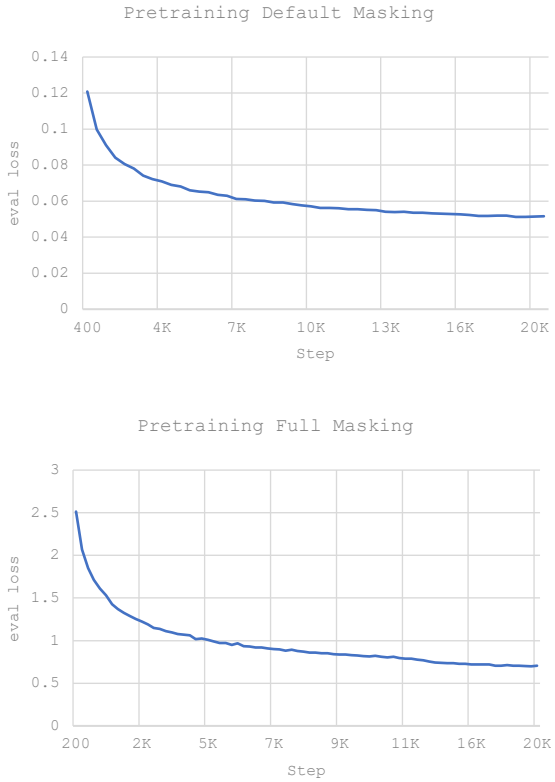

Figure 3: Cross Entropy loss curve for distilroberta pretrained on our dataset for 20k iterations.

Although the loss is lower for the default masking, an analysis of the predictions and input importance, which we will expose in the next section, indicated that the full masking was more adequate for our domain problem. Later analysis on fine tuning yilded the sames conclusions.

6.2. Fine Tunning

The next step consisted in finetuning both the pretrained model and a randomly initialized model on the atomic tasks of each event.

As it is obvious in Figure 4, the pretrained model has an advantage since its evaluation loss starts at 0.58 while the randomly initialized model starts at 0.85. This shows that the MLM task is beneficial to a downstream task of predicting atomic tasks of telemetry events, thus suggesting the hypothesis that using deep learning with a self-supervised pretraining could be helpful to learn different tasks. Further evaluation of the model's performance will confirm this hypothesis in the next section.

7. Evaluation

In this section, we will evaluate the quality of the model either for (1) the ability to learn patterns during pretaining, or (2) the ability to learn atomic tasks. We will look at implicit information that might be encoded in the embeddings using gradient saliency. Also regarding the learned embeddings, we will compare them with the Productivity Areas, by showing that they encode some of that information. We will also show the extrinsic evaluation of the fine-tuned model, using precision, recall and f1-score.

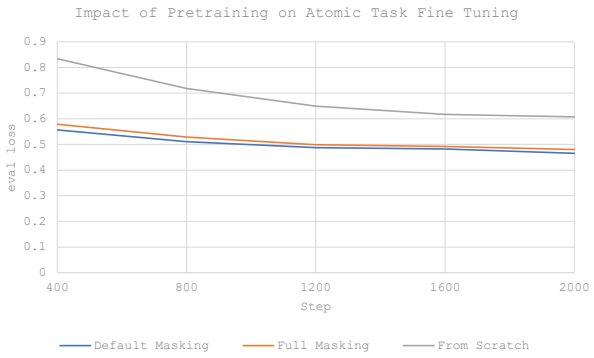

Figure 4: Cross Entropy loss curve for distilroberta pretrained on our dataset for 20k iterations.

7.1. Pretraining

To understand better what the model is learning we look at the gradient saliency of some examples. This enables us to understand (1) some examples where the model is failing; and (2) what parts of the input contribute the most for a specific prediction.

In Figure 5 we can see the model's input and its prediction. We can also see which tokens were most relevant for predicting the `Value` token. In this example, the model learns to reproduce the last event of a sequence almost exactly. If we hover over the predicted word `Value`, we can see it is attending to the whole context and probably not only remembering a simple rule from its immediate neighbors.

However, this is not true for the pretrained model on the default masking. In Figure 6, when the model is correctly predicting the token `TC` , from `LEFTCLICK`, we can see that the model is attending mainly to its immediate tokens to determine its value. This would be expected, since there are no more possibilities of log terms starting with `LEF` and ending with `LICK`. To test this hypothesis we ran the model through the full event masking (which is almost compatible with a highly unlikely default masking) on this same input, and the results were disastrous (see Figure 7). The full event masking reliably performed better qualitatively on almost all the examples we analyzed, which shows that the full masking strategy was the best to our goals. From now on, the model we will be analyz-

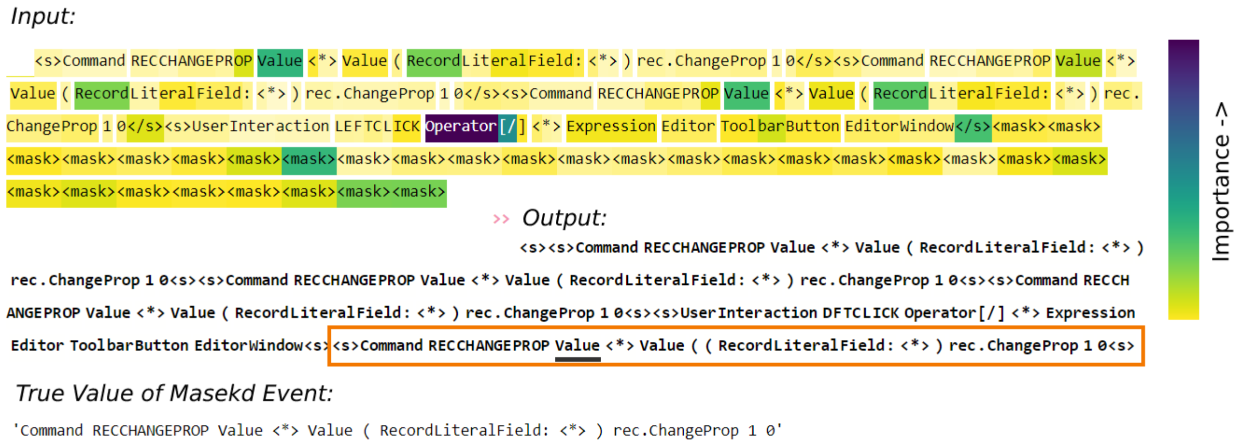

Figure 5: The gradient saliency visualization from Ecco applied to a sample of our dataset (with fewer concatenated events for visual simplicity). On the top, the input of the model with a full event masked. On the middle the output predicted by the model, with the masked event highligted in orange. On the bottom, the true value of the masked event.

ing is the one pretrained with FEM.

In Figure 8, the model learns that, in order for the user to get to the event `SystemEvent SELECT PropertiesPresenter OnBoarding_InsertPhoneNumber NRNodes.WebBlock Block OnBoarding_InsertPhoneNumber` , the user needs to make a UserInteraction by left clicking in the ESpace Tree Tabs ToolbarButton.

In Figure 9, two events were masked, and the model learns to copy a variable name defined by the user. This is interesting, because the model seems to have learned what is a custom named variable, and the fact that it usually appears multiple times in a patterned way. Highlighting the gradient saliency of the first token of the variable, we can see that the model is attending to both appearances of the same variable, but more to the first one. It is also worth noting that the model predicted two full consecutive masked events almost perfectly.

In Figure 10, there is also a custom named variable, but it is in portuguese. The model, here

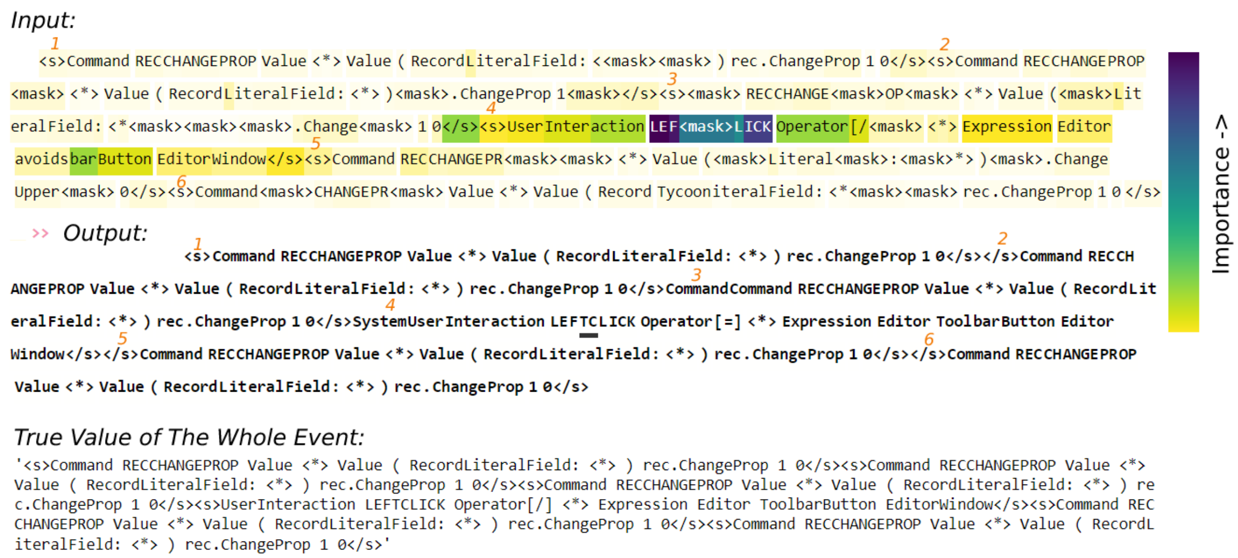

Figure 6: An example of an input masked in the original BERT approach. This leads to an overall better loss, but analyzing it qualitatively we can see that the model is focusing more on the immediate neighbors of the masked token TC, and failing simple separator tokens. The numbers 1-6 indicate the beginning of each event, to help the comparison between input and output.

Figure 7: The same pretrained model with the default masking, but predicting an event with full event masking applied. It is obvious that the model cannot predict this event reliably, because each individual masked token does not have immediate neighbors that determine its value.

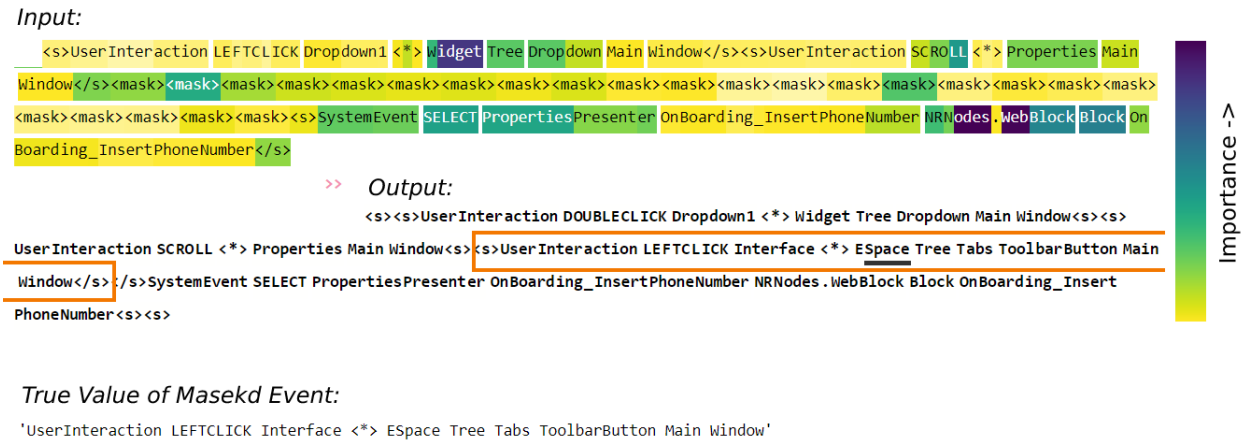

Figure 8: Second example of a success of our model.

struggles to reproduce that name. By looking at the saliency and the prediction, it is reasonable to say that the model is mixing two portuguese terms ”ListaEntidades” and ”ListaCampos” into ”Listaendosos”.

Figure 17 shows the result of trying to prompt engineer a failed example to work. The event `UserInteraction LEFTCLICK SelectedAggregator changed by User ( Legacy Event ) <*> Tabs` was masked and all the next events removed. This yielded a wrong prediction: `UserInteraction LEFTCLICK AddRemoveReferencesFromToolBar by*> Main Tool ) <*Button Main Window`. In Figure 17, we added back one event after the masked one, to give some more context to the model, and then it predicted it perfectly. We don't show all the iterations of this example to keep this section brief, but more examples can be found in Annex TODO

7.1.1 Fine Tuning

For obtaining the extrinsic performance metrics on the atomic tasks, we classified 350k datapoints, which corresponded to 11k individual events/atomic tasks. The class distribution is not uniform within each datapoint, so, each metric was not evaluated for the same number of events. Also, some events are very rare, which makes it difficult to find examples in 350k randomly sampled datapoints. This fact suggests that we should have grouped sporadic atomic tasks into an ”Other”

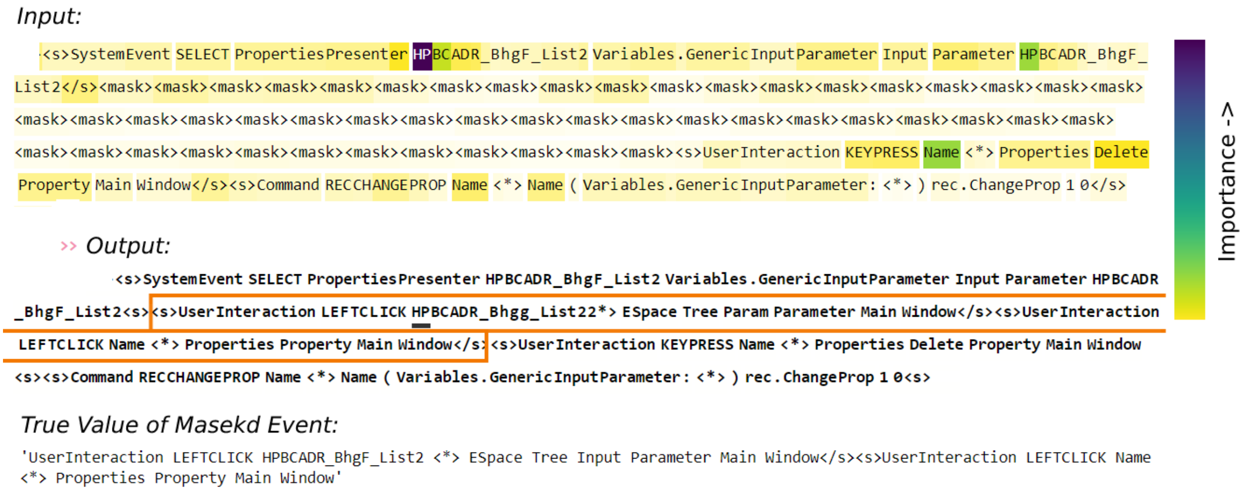

Figure 9: Third example.

Figure 10: Fourth example.

bucket to increase its volume. As it is, the model probably did not have many examples of some of these classes, although the results do not suggest this for almost all of them (full results in Annex B). Improving these methodological imperfections is left for future work. However, it is important to stress that our goal with this Fine Tuning section was not to optimize the fine tuning stage of our pre-trained model, but to show that the trained embeddings contain useful information for different downstream tasks.

In Figure 12, we can see that the model is able to learn most atomic tasks, obtaining a weighted precision, recall and f1 score of 87% (as it is shown in Figure 13). Figure 18 only shows atomic tasks that had a volume larger than 500 atomic tasks. A full table can be found in Annex We can also see that our hypothesis of the default BERT masking not being appropriate for our use case was correct, since it does not incentivize the model to learn relationships between events, but within each event, as we showed previously. However, the degree to which the hypothesis was correct is unexpected, since the default masking provided a model completely unable to learn any atomic task.

Figure 11: Fifth example.

	precision	recall	f1-score	support
ChangeProperty	93%	90%	91%	155,154
Navigate	84%	90%	87%	133,396
Debug	95%	91%	93%	26,100
1CP	84%	79%	81%	15,794
DeleteObject	88%	75%	81%	12,600
ManageDependencies	73%	86%	79%	5,218
AddConnector	82%	80%	81%	4,249
MoveWidget	88%	81%	84%	3,644
AddNode	88%	72%	79%	3,019
CopyNode	81%	69%	74%	1,234
AddInputParameter	87%	77%	81%	1,106
MoveNodeInFlow	61%	73%	67%	941
AddAttribute	79%	84%	81%	875
AddLocalVariable	85%	69%	76%	818
AddClientAction	79%	87%	83%	758

Figure 12: The performance metrics of the model pretrained with full event masking, fine tuned for 40k steps in the atomic tasks goal.

Therefore, in order to understand if there was some information regrading the productivity areas encoded in the embeddings, we decided to train a Random Forest on the embeddings for predicting the corresponding PA. The idea is that, if the random forest can predict the PAs from the embeddings, there is at least some information encoded regarding the PA (i.e. some information was encoded regarding the UI of the software). The results of this simple test can be found in Figure 14.

These embeddings correspond to the embeddings of each [SEP] (or <s> in case of Roberta) token. For each [SEP] there is a label of a Productivity Area.

We can see that the model was able to learn the most common productivity areas (Interface, Logic and Data) but was unable to learn the categories with low volume, or we did not evaluate enough samples to have a significant result in those cases. However, again, our goal here was not to optimize this task, but to show that some of the encoded information is correlated with the productivity areas. This is promising for future work, as our pretraining seems to introduce general useful information for different tasks.

8. Discussion and Future Work

Our approach of training a transformer-based model on Telemetry data to learn its patterns and distribution was successful. The BERT-like architecture was able to learn the data in a self-supervised way. As future work adding more data fields that we left out (e.g. CUSTOM_DATA) or timestamps and the duration of each event can be helpful for the model to learn different representations. For some use cases, an encoder-decoder (or just a decoder architecture like GPT3) can be tried. For example, if we are interested in predicting the next steps of a user. Yet another idea for future work regarding the initial setup of our problem would be to use a model that allows for dynamic or bigger input sizes (like Longformer) or to preprocess the dataset with different random sizes (to force the model to learn stronger relations with its immediate neighbors).

The methodology we chose to arrive at a pretrained model that would best fit our needs (i.e. distilRoberta) seemed to provide a good model that achieved our goals within our time constraints. As future work, it would also be interesting to let the other competing architectures pretrain for a few more steps to see if any model shows some hidden potential. For example, one initial hypothesis

	Precision			Recall			f1-score		
	Weighted Avg	Macro Avg	Micro Avg	Weighted Avg	Macro avg	Micro avg	Weighted Avg	Macro avg	Micro avg
Default Masking (20k)	0.7%	0.2%	3.2%	3.2%	1.6%	3.2%	1.2%	0.3%	3.2%
Full Event Masking (20k)	87.3%	50.9%	87.2%	87.2%	45.7%	87.2%	87.1%	47.2%	87.2%
Full Event Masking (40k)	88.4%	67.2%	88.2%	88.2%	67.6%	88.2%	88.2%	66.0%	88.2%

Figure 13: The performance metrics of the two tokenization strategies in the atomic tasks goal.

Productivity Area	Precision	Recall	F1-score	support
Application Management	100%	33%	50%	218
Data	93%	47%	62%	23,297
Forge	0%	0%	0%	5
Interface	78%	90%	83%	143,659
Logic	72%	65%	68%	79,824
Module	0%	0%	0%	321
None	0%	0%	0%	21
Other	100%	23%	38%	194
Process	100%	1%	2%	1,351
Unknown	100%	2%	4%	364
idle	100%	1%	1%	746
macro	67%	24%	28%	250,000
weighted	77%	77%	75%	250,000

Figure 14: The performance metrics of the random forest trained on the output embeddings of the pretrained model.

we had was that a model trained on coding data could be more suitable to our dataset, since the gist of Service Studio is to abstract common programming patterns.

Our gradient saliency analysis showed that our custom full event masking was crucial for the success of this thesis. The default masking from BERT made the model learn the internal structure of each isolated event, instead of the relationships between events. Although the loss of the former was lower, the qualitative analysis showed that the model was not learning what we pretended. Further analysis of the gradient saliency of some examples showed some inputs where the model struggled to predict the correct event, and some where the model predicted perfectly. Looking at some of these examples, it is clear that some examples are probably impossible to predict (after all it is a human being that is using the Service Studio) and some that would probably improve with further training.

After the pretraining, we applied the learned representation of the events, decomposed in subwords, by fine-tuning the model to learn the atomic tasks dataset. This was also a success, which showed (1) an application of the learn embeddings, (2) one proof that the MLM task provides embeddings general enough to serve a different task, which was not the case for the default BERT masking for example.

Finally, as way to extrinsically evaluate the learned embeddings during pretraining, we trained a Random Forest on those embeddings to learn a third dataset: Productivity Areas. In this step is became clear that the learned embeddings also contain some general information regarding the structure of Service Studio, since the Productivity Areas refer to UX/UI elements of the software. We used the [SEP] token to represent the whole event, but different approaches may be tried in future work, like averaging all the tokens of the event.

Another step left for future work is to visualize these learnt embeddings using dimensionality reduction techniques. We began this process using PCA but were unable to finish due to time constraints, but the initial result showed no discernable patterns. Different dimentionality reduction strategies like UMAP ou T-SNE an be tried to yield different insights on what kinds of information the model is learning. Also, help of experts that deeply know how Service Studio works can help in this interpretability problem.

In the end, we showed that our model is a good general approach to tackle problems like the ones we suggested in the Dataset chapter, such as Stuck Moments, Productivity Areas, Next Event Suggestion, etc. It is now reasonable to say that this approach is a good contender for any deep learning problem that requires telemetry data, since the can probably benefit from these embeddings as input features.

References

[1]

[2] X. Chen, Y. Wu, Z. Wang, S. Liu, and J. Li. Developing real-time streaming transformer transducer for speech recognition on large-scale dataset. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5904–5908. IEEE, 2021.

[3] L. Dong, S. Xu, and B. Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on*

Acoustics, Speech and Signal Processing (ICASSP), pages 5884–5888. IEEE, 2018.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] C. Han, M. Wang, H. Ji, and L. Li. Learning shared semantic space for speech-to-text translation. *arXiv preprint arXiv:2105.03095*, 2021.

[6] P. He, J. Gao, and W. Chen. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing, 2021.

[7] Y. Jiang, S. Chang, and Z. Wang. Transgan: Two pure transformers can make one strong gan, and that can scale up. *Advances in Neural Information Processing Systems*, 34, 2021.

[8] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[9] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[10] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu. Neural speech synthesis with transformer network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6706–6713, 2019.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[12] L. Michael. Explainability and the fourth ai revolution. 2021.

[13] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[14] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[15] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid. nd] b. videobert: A joint model for video and language representation learning. in 2019 ieee. In *CVF International Conference on Computer Vision (ICCV). IEEE*.

[16] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[18] A. Vedaldi, H. Bischof, T. Brox, and J. Frahm. Springer: Cham, 2020.

[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[20] H. Young, V. Dumoulin, P. S. Castro, J. Engel, and C.-Z. A. Huang. Composing features: Compositional model augmentation for steerability of music transformers. 2021.

[21] J. Zhang, Y. Zhao, M. Saleh, and P. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

[22] Y. Zheng, X. Li, F. Xie, and L. Lu. Improving end-to-end speech synthesis with local recurrent neural network enhanced transformer. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6734–6738. IEEE, 2020.

A. Pretaining Gradient Saliency full example

Figure 15: Fifth example.

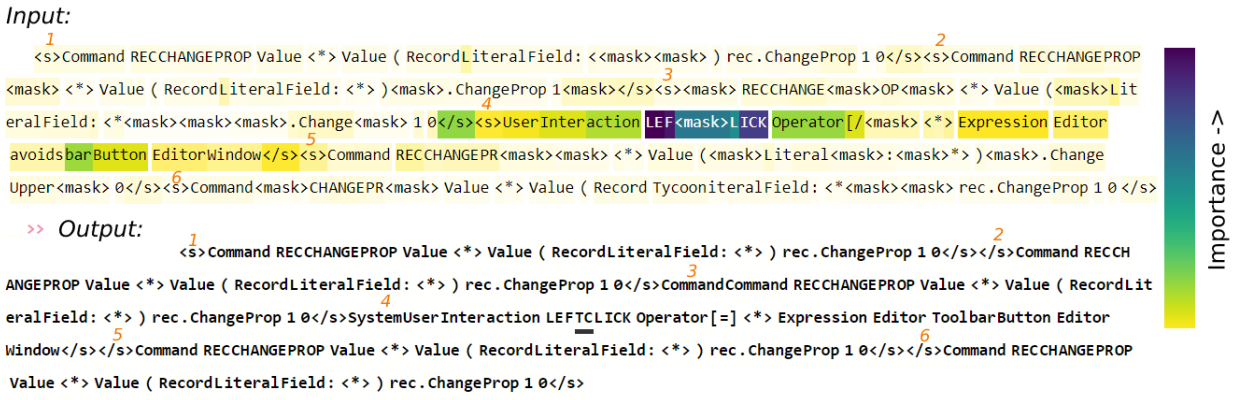

Figure 16: Fifth example.

Figure 17: Fifth example.

B. Fine Tuning Results

	precision	recall	f1-score	support
ChangeProperty	93%	90%	91%	155,154
Navigate	84%	90%	87%	133,396
Debug	95%	91%	93%	26,100
1CP	84%	79%	81%	15,794
DeleteObject	88%	75%	81%	12,600
ManageDependencies	73%	86%	79%	5,218
AddConnector	82%	80%	81%	4,249
MoveWidget	88%	81%	84%	3,644
AddNode	88%	72%	79%	3,019
CopyNode	81%	69%	74%	1,234
AddInputParameter	87%	77%	81%	1,106
MoveNodeInFlow	61%	73%	67%	941
AddAttribute	79%	84%	81%	875
AddLocalVariable	85%	69%	76%	818
AddClientAction	79%	87%	83%	758
AIPick	99%	75%	85%	495
AddRecord	82%	86%	84%	464
AddServerAction	72%	84%	77%	438
AddScreenAggregate	95%	71%	81%	437
AddSort	85%	76%	80%	260
AddScreen	87%	82%	84%	231
AddStructure	86%	80%	83%	205
AddSource	84%	69%	76%	202
AddDataFromOtherSource	70%	84%	76%	185
AddJoin	67%	82%	74%	165
AddEntity	88%	80%	84%	157
AddWidget	72%	94%	81%	150
AddConsumeREST	100%	53%	69%	132
AddBlock	68%	86%	76%	129
AddModule	85%	93%	89%	127
InstallAppForge	75%	21%	33%	85
AddStaticEntity	86%	78%	82%	72
AddActiontoBoostrapDataFromExcel	66%	99%	79%	68
AddFilter	0%	0%	0%	64
AddProcess	92%	88%	90%	51
AddRole	79%	75%	77%	44
AddTimers	98%	100%	99%	43
AddResource	100%	93%	96%	43
AddConsumeSOAP	83%	97%	89%	34
ImportNewEntitiesFromExcel	75%	95%	84%	22
AddOutputParameter	77%	50%	61%	20
AddScript	78%	100%	88%	18
LoginEnvironment	23%	59%	33%	17

Figure 18: The performance metrics of the model pretrained with full event masking, fine tuned for 40k steps in the atomic tasks goal. Even tasks with very low volume show impressive results