POLITECNICO DI TORINO

EURECOM

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Personal Data Detection in Free Text

Supervisors:

Dr. Francesco Di Cerbo

Prof. Paolo Papotti

Prof. Giuseppe Rizzo

Candidate:

Gabriele Gioetto

Academic Year 2022/2023
Torino, Sophia Antipolis

Abstract

An HR (Human Resources) department in a large organization receives inquiries/requests from employees on multiple topics, which are quite different from one another. As an example, an employee can send requests dealing with health conditions, compensation/taxation, events of life (marriage, death of a relative...)....
These data can be used for many different queries that can be useful for analysis purposes (Example: 'How many people have had COVID during 2021'). However, HR tickets typically contain personal data, that cannot be processed without the consent of the data subject according to the European privacy regulation (GDPR).
To be able to process documents with personal data, we can identify the pieces of information that qualify as personal data in a communication and subsequently anonymize such information using the appropriate techniques. A significant part of this problem is represented by the complex nature of personal data according to GDPR: personal data are defined as '*any piece of information that can be connected to an identified or identifiable natural person*' It comprises obvious identifiers like social security numbers, email addresses but also, elements like 'the Italian intern working for SAP in South of France'. To the best of our knowledge, it does not exist a public dataset of HR tickets which can be used to train machine learning models, the main reason being the difficult nature of these types of data. Synthetic data, which are artificial data that are generated from original data and a model that is trained to reproduce the characteristics and structure of the original data, follow a data protection by design approach. To address the need for a large dataset of HR tickets, we created a taxonomy of tickets, we found real data that can be used as support to create synthetic tickets and developed Ticket Generator: an application that can produce as many tickets as needed

belonging to different categories, we released a dataset of previously created tickets and we showcase some possible use cases of the dataset.

Acknowledgements

About SAP

Founded in 1972, SAP has grown to become the world's leading provider of business software solutions. SAP is market leader in enterprise application software. The company is also the fastest-growing major database company. Globally, more than 77% of all business transactions worldwide touch an SAP software system.

With more than 347.000 customers in more than 180 countries, SAP includes subsidiaries in all major countries. SAP is the world's largest inter-enterprise software company and the world's third-largest independent software supplier, overall. SAP solutions help enterprises of all sizes around the world to improve customer relationships, enhance partner collaboration and create efficiencies across their supply chains and business operations. SAP employs more than 98.600 people.

Security Research at SAP Labs France, Sophia Antipolis

Based at SAP Labs France Mougins, Security Research Sophia-Antipolis[1] addresses the upcoming security needs, focusing on increased automation of the security life cycle and on providing innovative solutions for the security challenges in networked businesses, including cloud, services and mobile. This internship is based in the SAP Labs France Research Lab, in Sophia-Antipolis. The work will be performed in the context of the Research Program "Security & Trust".

Table of Contents

Security Research at SAP Labs France, Sophia Antipolis v

List of Figures IX

Acronyms XII

1 Introduction 1

1.1 GDPR 1

1.2 Synthetic data 2

1.3 Ticket Generation 3

2 Related Works 4

3 Method 6

3.1 Taxonomy 6

3.2 Datasets 8

3.2.1 Datasets preprocessing 9

3.2.2 Bayesian Network 10

3.3 Ticket Generation 12
3.4 State of The Art auto-regressive language model 19
3.5 GPT-J 21
3.6 Survey 30
3.7 Results 31
3.8 Additional Experiments 39
3.8.1 Topical Generation 39

4 Use Cases 42
4.1 Classification 42
4.1.1 FastText Classifier 43
4.1.2 BERT classifier 44
4.1.3 Results 45
4.2 Anonymization 47
4.3 Named Entity Recognition 53
4.3.1 Weak labeling 54
4.3.2 Classical NER 57
4.3.3 Our approach to NER 58

Bibliography 62

List of Figures

3.1 Initial Taxonomy 6

3.2 Final taxonomy, reduced due to unavailability of data 7

3.3 Schema of Ticket Generation 14

3.4 Input Saliency: First token 17

3.5 Input Saliency: Subject 17

3.6 Input Saliency: HR 18

3.7 Neuron Activation Analysis 19

3.8 Non-negative Matrix Factorization on Activation Matrix . . 19

3.9 GPT-3 and GPT-J architectures compared 27

3.10 Byte-Pair Encoding Tokenization Example 28

3.11 Implementation of RoPE, Image taken from original paper . 30

3.12 Example of survey 32

3.13 Hierarchy of all topics generated by BERTopic 38

3.14 Main 8 topics generated by BERTopic 38

4.1 Schema of BERT classification 45

4.2 Confusion matrix of Ticket classification with BERT 47
4.3 Schema of Ticket Anonymization 50
4.4 Schema of QAGS 53
4.5 Schema of SUMMAQA 54
4.6 Schema of Cosine Similarity for 55
4.7 Ticket NER preprocessing part 1 60
4.8 Ticket NER preprocessing part 2 60
4.9 Ticket NER training 61

Acronyms

Chapter 1

Introduction

With the advent of deep learning a lot of applications have been needing more and more data. . However, more often than not, these data contain a large amount of sensitive and personal information that restricts its use according to the legal framework in place in many countries.
Even in companies' internal data, massive amounts of sensitive information are growing, limiting its processing and sharing. It is considered that, if the information reveals the identity of a person then it threatens the personal rights of this person, and can only be processed with special attention in compliance with the legal framework.
This is especially relevant in the context of HR tickets, which can contain not only personal information but also special categories of personal data, which need additional layers of protection according to GDPR.

1.1 GDPR

The General Data Protection Regulation is a privacy regulation that regulates the processing of personal data "wholly or partly by automated means and to the processing other than by automated means". The regulation applies to all citizens of the EU and to all data subjects in the EU, whether the processing is carried out inside or outside the EU.

Personal data are defined as "any information relating to an identified or identifiable natural person". The regulation states that "an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person". The natural person can be identified both by direct identifiers and quasi-identifiers. The direct identifiers are information that directly identifies the person, such as the telephone number, the social security number..., while the quasi-identifiers are information that alone cannot identify a person, but if they are combined with other quasi-identifiers can affect a person's privacy. For example, the job title could be not enough to identify a person, but combined with his/her company and his/her nationality could be.
Moreover, the GDPR treats some categories of personal data more carefully. These categories are called 'special categories' and include racial or ethnic origin, political opinions, religious or philosophical beliefs, trade union membership, genetic data, biometric data, data concerning health or data concerning a natural person's sex life or sexual orientation
The special categories of personal data cannot be generally processed, with some exceptions including "the data subject has given explicit consent to the processing of those personal data for one or more specified purposes".

1.2 Synthetic data

Synthetic data is artificial data that is generated from real data and has the same statistical distribution as the original data. This means that synthetic data and original data should deliver very similar results when undergoing the same statistical analysis.
Synthetic data has many benefits over real data: if you create a model that generates synthetic data you can generate how many data you need, you can infer certain properties to your data (for example it can be useful for bias and fairness research) and above all, synthetic data can respect the right to personal data protection. However, it is not always guaranteed that synthetic data is privacy-preserving: it has been shown that synthetic data

can leak personal information [2].

1.3 Ticket Generation

We created a tool that can be used to create an unlimited amount of synthetic HR tickets and we published a dataset of 16000 tickets. Each ticket, other than the ticket's text, is composed of a category, sub-category and the entities of the ticket. The tickets are not created from scratch, but starting from a dataset of real data and some prompts that help the model generate the text.
We did a survey internal to the company to gather some real data, and we changed the ticket generation parameters in order to respect the real data with respect to different text metrics.
Finally, we showed different possible use cases for the dataset:

- Ticket Anonymization
- Ticket Classification
- Named Entity Recognition on tickets' entities
- Ticket Summarization

Chapter 2

Related Works

As far as we know, it has never been published a dataset of HR tickets written by company employees, due to the sensitive nature of the data. We believe this is the case not only because of the GDPR laws, but also because these data can be damaging both for the company and the employees in some cases (For example an employee criticizing the working environment or leaking some personal information in a ticket).
However, some datasets can resemble what we are trying to build, with similar language style and intents:

- *Consumer Complaint*: Consumer Financial Protection Bureau's online database of customer complaints about different financial products. The overall dataset contains over 600,000 complaints and each record has the complaint's text description, the product that is the cause of the complaint, the company which issues the product, and the category of the issue. In the dataset the personal information are masked.
- *Enron Mail*: The *Enron Mail* dataset contains about 0.5M emails of 150 senior management from the Enron corporation. This data was originally made public, and posted to the web, by the Federal Energy Regulatory Commission during its investigation for fraud. The corpus is one of the only few publicly available mass collections of real emails easily available for study.
- *TAB*: the Text Anonymization Benchmark corpus[3] is a privacy-oriented

annotated text resource. The corpus comprises 1,268 English-language court cases from the European Court of Human Rights (ECHR) enriched with comprehensive annotations about the personal information appearing in each document, including their semantic category, identifier type, confidential attributes, and co-reference relations.

Chapter 3

Method

3.1 Taxonomy

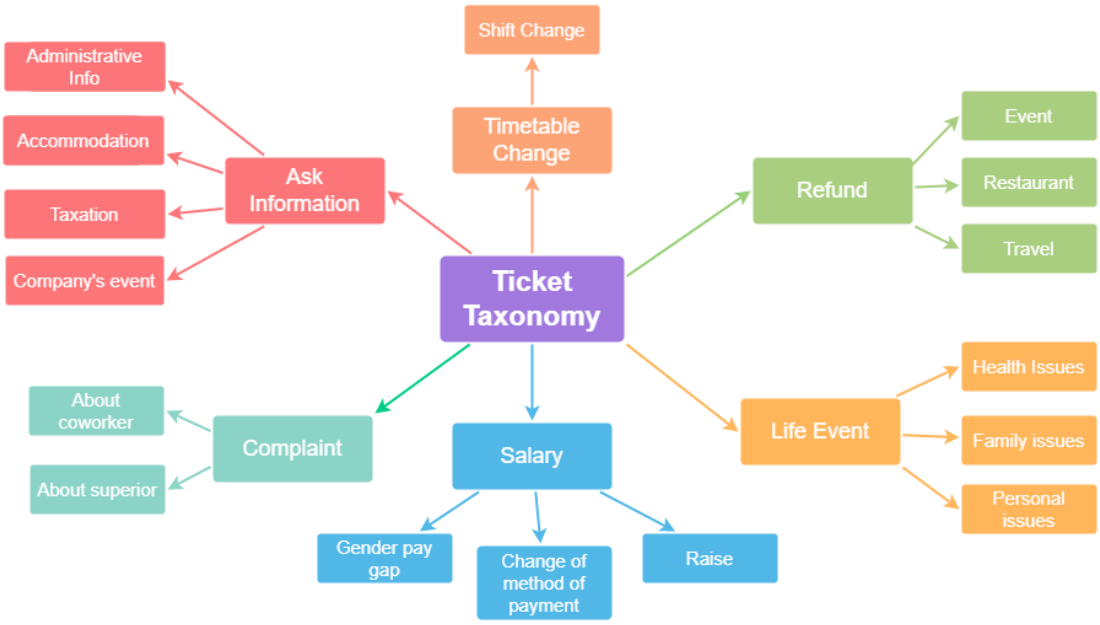


Figure 3.1: Initial Taxonomy

Usually, HR tickets can belong to various categories, which can range from a request for a shift change to a complaint about a superior.

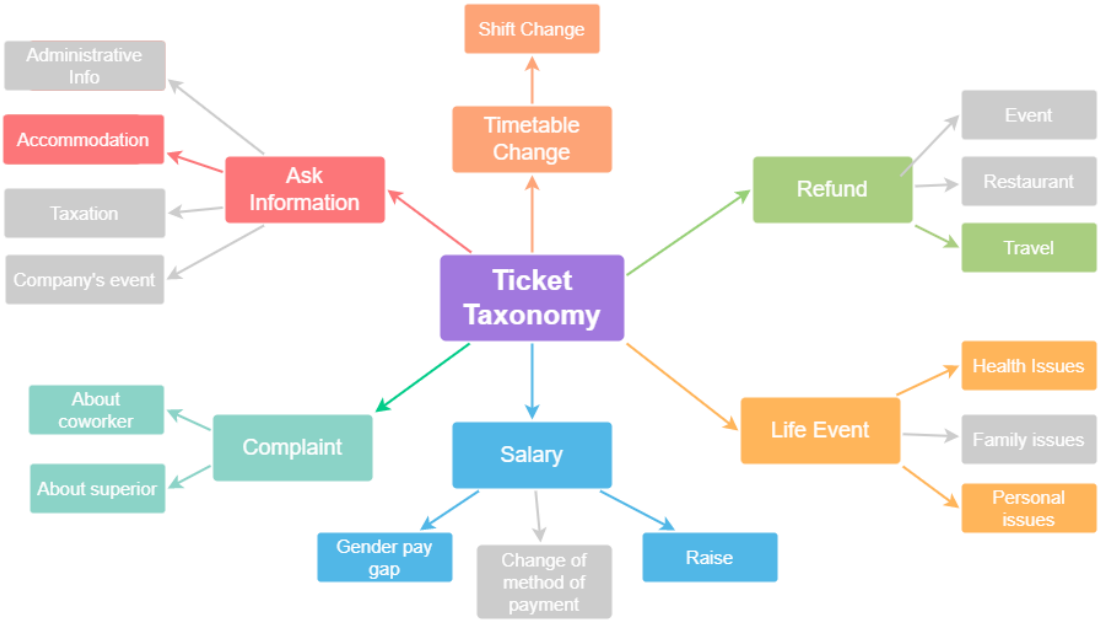


Figure 3.2: Final taxonomy, reduced due to unavailability of data

We built a taxonomy of tickets, which is structured into categories and subcategories. Each subcategory has its own variables that are used as inputs for ticket generation. The variables are sampled from the existing datasets mentioned before.
For example, to create a request for sick leave, we pass as inputs the reason and the number of days of sick leave requested. Moreover, each category has distinct templates and prompts. The taxonomy (Shown in Figure 3.1) has been built following the advice of an HR expert from SAP, however, the final taxonomy (Shown in Figure 3.2) presented here is a subset of the original one due to the unavailability of public data on certain topics (Ex. *Work benefits*)
The final complete taxonomy and the complete list of all variables used for each category/sub-category can be seen in the Table 3.1

Category	Sub-category	variables
Ask Information	Accommodation	location, duration
Complaint	About coworker	complaint, reason
	About superior	complaint, reason
Timetable change	Shift change	reason_of_change, old_date, new_date
Salary	Salary raise	old_salary, new_salary, increase,work_title
	Gender pay gap	wage_gap
Life Event	Health issues	disease, number_of_days_of_sick_leave
	Personal issues	issue, number_of_days
Refund	Travel	from, to, date_travel

Table 3.1: Table of all defined categories and sub-categories with their respective variables

3.2 Datasets

In order to generate tickets, we decided to use real data as a starting point to make them as much realistic as possible. Another reason to use real data is that it makes the dataset useful for use cases such as anonymization.
The datasets that we used are all public and available online. In some cases where no datasets were available, we created them manually from personal experience.
The datasets are:

- *Absenteeism at work Data Set*: contains records of work absences, with the reason for the absence (almost always a disease) and the number of hours of absence. It is used to create requests for days off for health reasons
- *National Occupational Employment and Wage Estimates United States*: estimates of wages in the US calculated with data collected from employers in all industry sectors in metropolitan and nonmetropolitan areas in every state and the District of Columbia. It is used to create requests for salary raises.
- *List of events of life*: list of major events in life. It is used to create requests for time off due to personal reasons.
- *Gender pay gap in the UK*: dataset of employers with 250 or more

employees, comparing men's and women's average pay across the organizations. It is used to create the requests for explanations for the wage gap between genders.

- *OpenFlights database*: datasets of airports and flights all over the world. It is used for requests for refunds of travel.

- *Geonames all cities with a population over 1000*: datasets of all cities of the world with a population over 1000 people. It is used for requests for information about accommodation.

3.2.1 Datasets preprocessing

The *Absenteeism at work Data Set* is the only dataset that contains data about people that is not already grouped and averaged. This means that there is a record in the dataset for each employee request, which contains the personal information of the employee, the reason for the absence and the time of absence in hours. For privacy reasons, we used a Bayesian Network. A Bayesian network is a probabilistic graphical model that measures the conditional dependence structure of a set of random variables based on the Bayes theorem. The features that we have used to build the Bayesian network are the *reason for absence*, the *month of absence* and the *time of absence*.
Using the *Absenteeism at work Data Set* we learn conditional probability distributions from data, to which we add a Laplace noise for differential privacy. Then we can sample new data that follow the original distributions, but that are not equal to the original ones.
The absence reasons in the dataset are given as ICD(International Classification of Diseases) codes, to make them more human-readable, we picked for each ICD code the corresponding more frequent diseases.
In the *National Occupational Employment and Wage Estimates United States* dataset, we sample employees based on the number of people employed in a certain field. Therefore for example since 'Retail Sales Workers' consists of 5.4% of the total occupations, then the sampled employee will have the 5.4% of possibility to have as occupation 'Retail Sales Worker'.
The current salary of the employee is calculated by adding Gaussian noise to the average salary of the employee's occupation, and then the salary raise requested is picked randomly between 5%-10%.

The ranges of the wage gap, used in the tickets regarding the explanation for the gender wage gap in the company, are sampled randomly from the dataset *Gender pay gap in the UK*, adding a Gaussian noise for privacy reasons.
To sample the cities for the requests of accommodation, we randomly sample from all the cities with more than 100,000 inhabitants from the country of residence of the synthetic employee. To calculate the duration of the accommodation we pick a random number of months between 1 and 12.
To get data for the category type 'refund travel', we sample randomly one flight from all the flights leaving from the country of the synthetic employee. The data are taken from the *OpenFlights database*.
The complaints about coworkers and superiors and the life events that can affect the work life of a person were written by myself, using primarily personal experience and some help from internet blogs.

3.2.2 Bayesian Network

A Bayesian network is a machine learning method that combines a probabilistic graphical model with Bayesian inference to infer the likelihood of certain events or outcomes. It is used to find relationships between variables and to identify which variables are most influential in predicting a certain outcome or event.
The bayesian network is based on the Bayesian theorem:

$$p(A|\ B) = \frac{p(B|\ A)p(A)}{p(B)}$$

Formally, the Bayesian network is a directed acyclic graph G = (V,E) with

- A feature for each node i belonging to V
- A conditional probability distribution for each edge, so the edge from feature i to j represents $p(x_j|\ x_i)$

The base version of a Bayesian network works with discrete variables, however, some implementations consider also continuous variables [4]

Building a Bayesian network starting from the *Absenteeism at work Data Set* is relatively easy, we calculate the likelihood distribution $p(x_i|\ x_j) \forall x_i, x_j \in D$, where D is the set of features. As a prior, we used a Dirichlet distribution, mainly because it is the conjugate prior of the categorical distribution.
We then added pseudocounts to the observed counts in the data used to calculate $p(x_j|\ x_i)$. This technique is used to diminish the overfitting of data. The values we used for pseudocount is $\gamma = 1$.
Since we learn the conditional probability distribution from our data, the structure of the network or the conditional probabilities may therefore leak some information on an individual in the dataset. In order to provide strong privacy guarantees and minimize the re-identification risk, we leverage the notion of differential privacy: we perturb the data adding a noise sampled from a Laplace distribution

$$z \sim Laplace\left(0, \frac{2 \cdot n_{features}}{\gamma \cdot \epsilon}\right)$$

where ϵ is the privacy budget for differential privacy, which controls the anonymization level.
Differential privacy is a rigorous mathematical definition of privacy. An algorithm is said to be differentially private if by looking at the output of an algorithm A performed on a dataset X, one cannot tell whether any individual's data was included in the original dataset or not. In other words, a differentially private algorithm guarantees that its behavior hardly changes when a single individual joins or leaves the dataset. The mathematical definition of differential privacy is:

$$Pr[A(X) \in Z] \leq e^{\epsilon} \cdot Pr[A(X') \in Z]$$

where A is an algorithm and X' is a neighbour dataset of X. A dataset is a neighbor of another dataset if they differ by only one record.
Once the private Bayesian network is built, we can sample new values for all the nodes in the graph. These generated values will have the same distribution and preserve the consistency and statistical properties of the original dataset up to the noise addition which acts as a de-identification barrier.

3.3 Ticket Generation

For each HR ticket, we create a synthetic employee. For all tickets' categories, the employee has some common features: *name*, *first name*, *last name*, *nationality*, *country*, *email*, *company*, *company's email* and *ticket date*.
All these information are created exploiting the Python library *Faker*. The nationality and the company's country are selected from the extendible list { USA, Germany, Italy, Spain, France }. All other information are created accordingly to the country picked. So for example, if the country of birth of the employee is Italy, then the generated name will be Italian.
Then, once the employees are generated, the information specific to the ticket category, the ones created starting from the open datasets as mentioned before, are concatenated to the general information of the employees.
For each ticket category, there are distinct templates. In each template there is an initial part that contains the general information of the employee, such as name, surname, company..., then some prompts correlated to the category of the ticket and then the textual prompt.
Here are a couple of examples of templates:

Request for time off due to health reasons:

> From: ${email}
> To: ${company email}
> First name: ${first name}
> Last name: ${last name}
> Company: ${company}
> Date: ${ticket date}
> Ticket category: ${category}
> Ticket sub-category: ${sub category}
> Date start absence: ${date start absence}
> Reason absence: ${reason}
> Subject: Request for sick leave for ${number_of_days}
>
> Dear Sir/Madame, my name is ${name} and I work at ${company}. I am requesting <*generate*>. I hope <*generate*>.

Request for refund of travel:

> From: ${email}
> To: ${company email}
> First name: ${first name}
> Last name: ${last name}
> Company: ${company}
> Date: ${ticket date}
> Ticket category: ${category}
> Ticket sub-category: ${sub category}
> Date Travel: ${date_travel}
> From: ${airport_from}, ${from}
> Destination: ${airport_to}, ${to}
>
> Hello, my name is ${name}. I am writing this mail to ask for a refund for the travel <*generate*>

The variables are replaced with the features of the employee, whereas the <*generate*> are replaced with text generated by a generative model. The parts generated by the generative model are created recursively. This means that the first <generate> is replaced with text generated automatically using as prompt everything that precedes it. Then the second <generate> will have as a prompt the entire ticket, including the text generated previously by the model. The model is forced to generate some text, if no text is generated in an iteration, the process is repeated until the model gives a non-empty output.
A general schema of the generation is shown in Figure 3.3

Templates

The model takes in a prompt, or context, and generates text from it. The prompts typically take the form of a few sentences or a paraphrase, and the model generates sentences that fit the context of the prompt. By manipulating the prompt, users can generate text of various tones, topics, and styles.

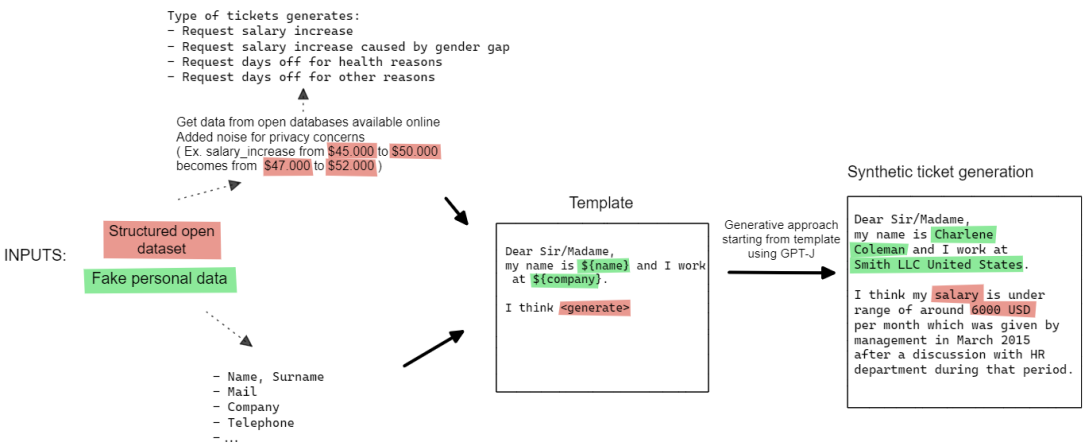


Figure 3.3: Schema of Ticket Generation

The GPT model is also capable of completing tasks such as question answering, machine translation, and summarization in an unsupervised manner[5]. By providing a prompt with the task and context, the model can generate accurate results that address the specific context and task. Tasks such as summarization, for instance, require a prompt to provide the model with the information it needs to summarize the text accurately.
Changing the prompt of GPT can change the tone, topics, and style of the generated text. Depending on the prompt, GPT can generate text ranging from creative stories to technical summaries. The tone and style of the text can range from humorous to academic, depending on the prompt. As the prompt changes, the model will also adjust to reflect the context of the prompt.
This is why we took inspiration from the e-mail format, as HR tickets are created in a working environment where formal language is used and often they resemble emails in the tone and the topics.
In particular, the Enron Emails dataset is included in the Pile dataset, the dataset on which GPT-J is trained.
The emails of the Enron dataset have usually a well-structured prompt, here's an example:

> Message-ID: <16593073.1075858228177.JavaMail.evans@thyme>
> Date: Wed, 12 Jan 2000 00:29:00 -0800 (PST)
> From: carrie.hollomon@enron.com

To: phillip.love@enron.com
Subject: Workhours
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Hello ...

To mimic the format of the emails of the Enron Dataset, we kept the From, To, Date and Subject rows. Instead, we removed the Mime-Version, the Content-type and Content-Transfer-Encoding, because after conducting some experiments it was evident that they did not help to achieve better results, on the contrary in some cases they were worse.
In addition to the standard information, we added explicitly some rows with additional information specific to the category, rather than including them only in the subject or in the initial prompt.

In order to generate tickets that were dissimilar and covered a wide range of topics/tokens, we preferred giving GPT small text prompts and letting the model generate most of the text getting the information from the email-like prompt. This approach allowed us to improve the diversity of the dataset and, consequently, its usefulness.

Small text prompt example of a request for shift change:

Dear Sir/Madame, my name is ${name}. I wanted to <*generate*>

Long text prompt example of a request for shift change:

Dear Sir/Madame, my name is ${name} and I work at ${company}. I wanted to ask to change the shift from ${old_date} to ${new_date} in order to be able to <*generate* >

Architecture analysis

To understand better how the model was behaving and why it gave certain types of outputs rather than others, we used the python library Ecco, which

creates interactive visualization that shows at which layers of the architecture the final token has been decided, which input tokens contribute the most for a prediction, ...
In particular, we used two different methods:

- Input Saliency: used to show how much did each input token contribute to producing the output token

- Neuron Activation Analysis: used to examine underlying patterns in neuron activations using non-negative matrix factorization

Input Saliency

To get a better grasp of the most useful information of the prompt, and to understand how we could modify it to achieve better results, we used Gradient * input (Shrikumar et. al). This technique calculates the partial derivatives of the output of the model and multiplies them with the input itself. Then the inputs with the highest scores are the ones that influenced the most the generation of the new tokens

$$score = x_i \bigtriangledown f(x_i)$$

where f is the architecture output.
The main problem with the Gradient * input method is that only one input is considered. Integrated Gradients solve this issue, computing the average gradient while the input varies along a linear path.

$$score = (x_i - x_i') \int_{\alpha=0}^{1} \bigtriangledown f(x' + \alpha(x - x'))\, d\alpha$$

Nevertheless, we used the Gradient * input method for computational issues (The Integrated Gradients took too much RAM of the GPUs).
In the images under, we underline some of the considerations we have done while defining the prompts.

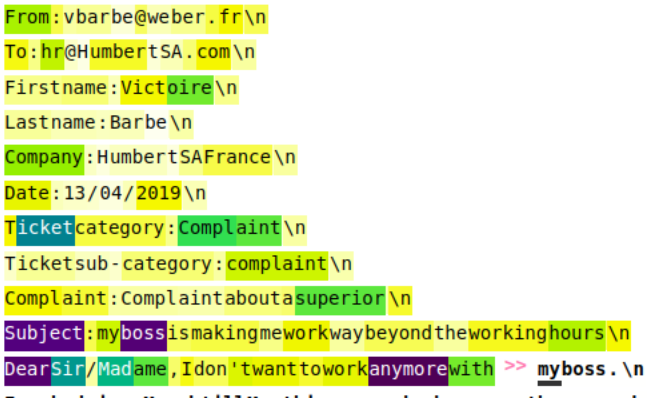


Figure 3.4: Input Saliency: First token

To create the first token, the model focus on the elements that resemble the content of an email, in particular on the fact that it is indeed a 'Ticket'

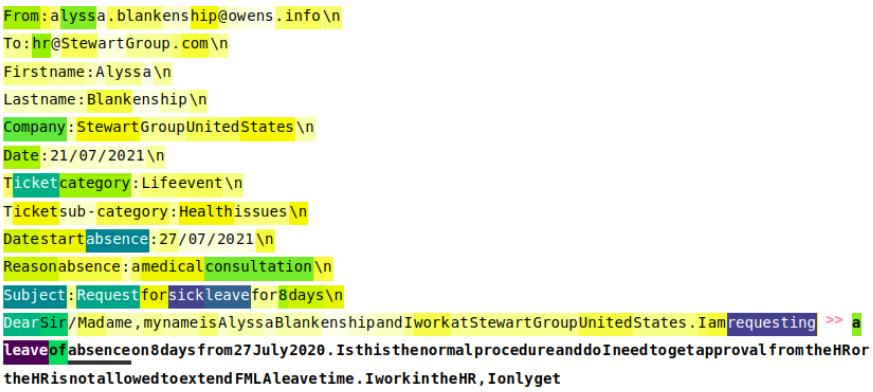


Figure 3.5: Input Saliency: Subject

Clearly stating the subject of the ticket helps the model create tokens that are on the right topic (In this example 'days of absence')

Neuron Activation Analysis

The Feed Forward Neural Network layer is one of the major components inside a transformer's block. To better understand how the neurons of different layers were 'activated' and how the neurons contributed towards each generated token, we exploited the Factor Analysis provided by Ecco.
Firstly, Ecco calculates the activation scores of each neuron over all layers, and then uses Non-negative Matrix Factorization3.8 to do dimensionality reduction on the matrix of activations, which will be reduced to a matrix

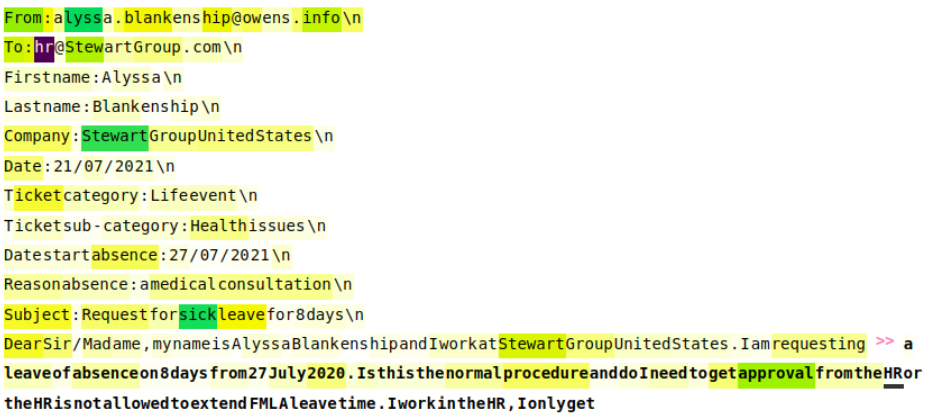


Figure 3.6: Input Saliency: HR

Clearly stating that we are writing to hr can help the model know the context, and consecutively use a certain language, specific words...

$M \times T$, where M is a parameter we decide and T is the number of tokens, starting from a matrix $(n \cdot N) \times T$, where n is the number of neurons per layer and N is the number of layers. In GPT-J $n = 16384$ and $N = 28$. Then, for each factor, which are the dimensionality-reduced layers, we visualize their activated neurons. From the example in the Figure 3.7, we can show for each factor what their main contributions are:

1. Punctuation
2. Mail prompt (From, To, First Name, Last Name, Company...)
3. New lines
4. Dates
5. First token, common across various GPT factors
6. Medical terms (usually terms related to the tickets' topic)
7. People's names and contacts
8. Company's name and contacts
9. Ticket's subject (Category, Sub-category and additional info)
10. Email presentation ('Dear Sir/Madame... ')

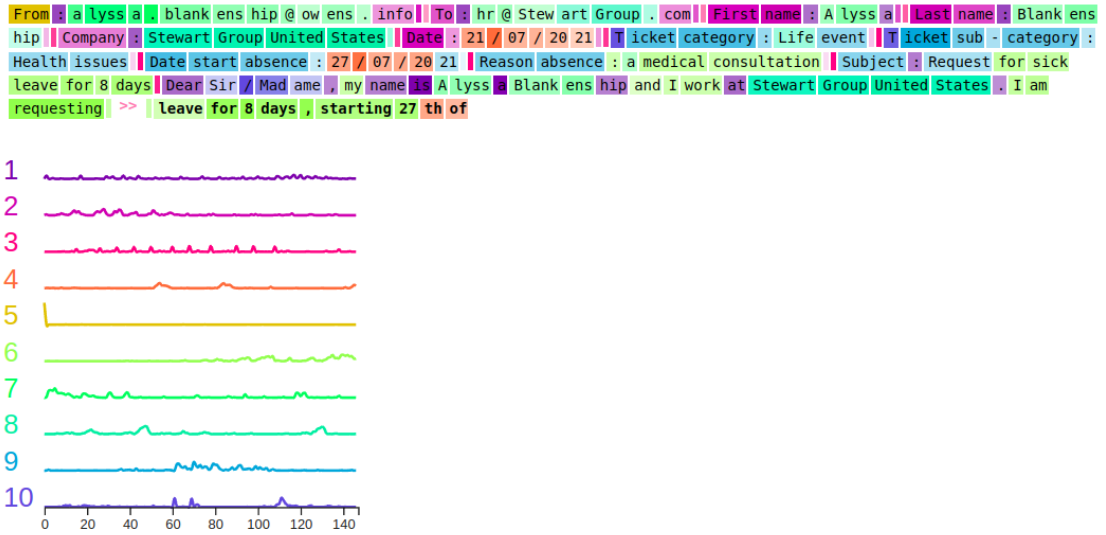


Figure 3.7: Neuron Activation Analysis

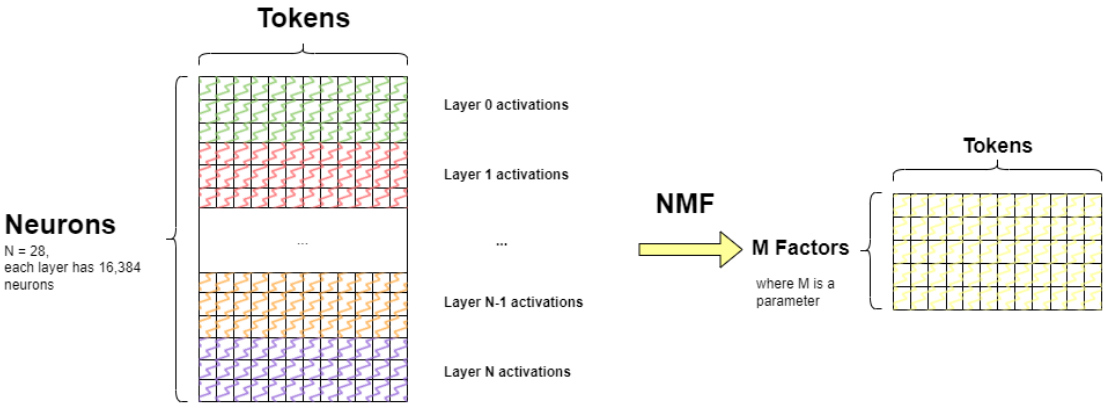


Figure 3.8: Non-negative Matrix Factorization on Activation Matrix

3.4 State of The Art auto-regressive language model

Autoregressive transformer-based Large Language Models are a sequence-to-sequence deep learning model that are pre-trained on a large corpus of data. They are designed to generate new tokens conditioning the model on some input text. The output probability distribution for the next token that the

model generates is a probability distribution over all possible tokens in the corpus.
Autoregressive transformer-based Large Language Models have shown ability to solve many tasks[6], varying from more classicalones like translation and question-answering to more peculiar one, like common sense reasoning and reading comprehension.
Mathematically, language models seek to maximize the likelihood of seeing some sentence by considering the product of conditional probabilities up to the point of generation

$$p(w_T|\ \theta) = \prod_{i=1}^{T} p(w_i|\ w_{1,..,i-1}, \theta) \tag{3.1}$$

Here we list some of the most popular auto-regressive language model:

- GPT-3: GPT-3 is an unsupervised AI language model developed by OpenAI. It is the successor to the previous iteration of GPT-2, and when released was the largest language model ever created. GPT-3 first showed that LLMs can be used for few-shot learning across different domains and can achieve state of the art results without building a task-specific model.

- PaLM: Pathways Language Model is a a 540-billion parameter LM developed by Google. The model was trained through the use of Pathways, a new system which enables highly efficient training of very large neural networks across thousands of TPUs. PaLM few-shot evaluation can outperform or match the finetuned state of the art on a wide array of reasoning tasks.

- Chinchilla: Chinchilla is a LM released by DeepMind. The peculiarity of Chinchilla is that it has 'only' 70B parameters, but outperforms models such as GPT-3 (175B). The authors of the paper demonstrate that for the same compute budget a smaller model trained on more data will perform better.

- GPT-J: GPT-J is an open-source alternative to OpenAI's GPT-3, the model was released by Eleuther AI, a group of independent researcher and has 6B parameters

- BLOOM: BLOOM is an open-source auto-regressive LM released by HuggingFace. BLOOM has a stronger focus on languages, differentiating itself from the majority of other LMs that mainly focus on the English language. In total, BLOOM was trained on 46 different languages

Amongst all of these LMs, we picked GPT-J, mainly because it was open-source, free and was not too big for our GPU memories. The other main alternative we have tried was GPT-2, the predecessor of GPT-3, which is now free-to-use. However, GPT-J's generated text was more coherent and more fluent

3.5 GPT-J

The generative model used to create the tickets is GPT-J, an open source 6 billion parameter, autoregressive text generation model trained on The Pile dataset released by EleutherAI.
The Pile dataset[7] is an 825 GiB English text corpus composed by 22 diverse subsets, which can be grouped in 5 categories:

- Academic (*ArXiv*, *PubMed Central*, ...)
- Internet (*Wikipedia*, *StackExchange*, ...)
- Prose (*Bibliotik*, ...)
- Dialogue (*Subtitles*, ...)
- Misc (*Github*, ...)

The parameters used for the generation of the next token are:

- *min_length*: minimum number of words created by a round of gpt generation

- *max_length*: maximum number of words created by a round of gpt generation.

- *top_k*: the k most likely next words are filtered and the probability mass is redistributed among only these k words. It helps the model not to go off-topic.

- *top_p*: the next words are sampled from the smallest possible set of words whose cumulative probability exceeds the probability p. Compared to *top_k*, in this case the size of the set of possible next token is not fixed. In practice, usually *top_k* and *top_p* are used together.

- *temperature*: the value T used to module the logits distribution. The higher the value of T, the higher the entropy of the logits distribution will be. In other words, tokens with an high probability will be less probable and tokens with a low probability will be more probable.

$$p_i = \frac{exp(x_i/T)}{\sum_j exp(x_j/T)}$$

- *repetition_penalty*: the parameter θ for repetition penalty. 1.0 means no penalty
  Given a list of generated tokens G,

$$p_i = \frac{exp\left(\frac{x_i}{T \cdot I(i \in G)}\right)}{\sum_j exp\left(\frac{x_j}{T \cdot I(j \in G)}\right)}$$

$$I(c) = \theta \text{ if } c \text{ is True else } 1$$

  So the logits distribution of the token change based on if the token has already been generated before.

- *length_penalty*: *length_penalty* > 0 promotes longer sequences, while *length_penalty* < 0 encourages shorter sequences.

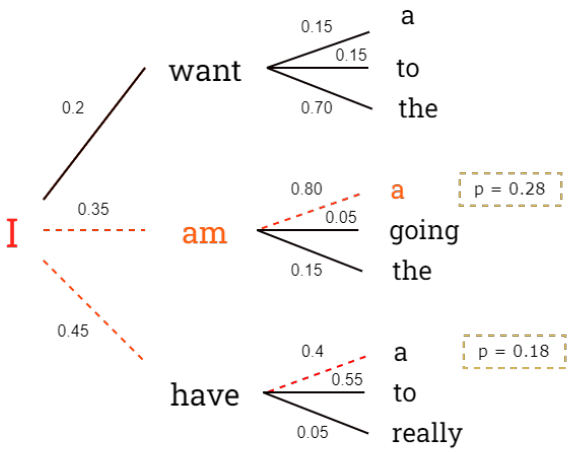
- *no_repeat_ngram_size*: If set to an integer > 0, all ngrams of that size can only occur once

Ex: $no\_repeat\_ngram\_size = 2$

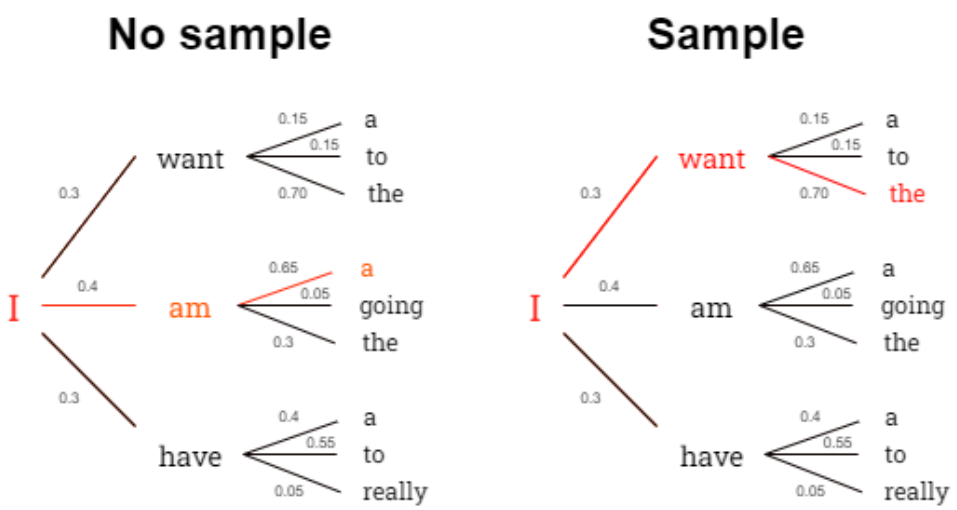
Dear Marie, I would like to have some days off. I would like to... ✗

Dear Marie, I would like to have some days off. I want... ✓

- *num_beams*: Number of beams for beam search. They beams are the number of 'paths' that are considered when choosing the next token. Even if a token is not the one with the highest probability, it can be chosen as the next token if the complete sentence is considered as more probable than the alternatives.



- *do_sample*: If set to 'False' greedy decoding is used (the most probable token is always chosen). Otherwise, sampling is used (the next token is chosen sampling from the distribution of possible next tokens)



- *bad_words*: List of words that are not allowed to be generated by the model.

- *force_words*: List of words that must be generated in the generation.

The default values assigned to the parameters used are shown in Table 3.2, however the application is set up in order to being able to apply different parameters at each run.

The GPT architecture is based on the original Transformers paper. The original architecture introduced two types of transformers blocks: the encoder block and the decoder block.
The encoder block converts an input sequence of tokens into a sequence of embedding vectors, whereas the decoder takes the output of the encoder and generate iteratively an output sequence of tokens.
GPT is pre-trained by predicting the next word based on the previous ones. GPT is decoder-only, which means that is assembled only by a stack of decoder blocks. Each decoder block is composed by:

- Normalization Layers: a normalization layer [8] normalize all inputs of a neural network across their features. It has been shown that Layer normalization enables smoother gradients, faster training, and better generalization accuracy [9]

Parameter	Value
min length	0
max length	50
top k	50
top p	0.85
repetition penalty	1.2
temperature	1
length penalty	1
no repeat ngram size	0
num beams	1
do sample	True
bad words	[ ]
force words	[ ]

Table 3.2: Parameters of GPT-J model

x: data sample
d: dimension of data sample
y: output of LayerNorm
ϵ: small number added for stability

$$u = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x\_ui)^2$$

$$\hat{x_i} = \frac{x_i - u}{\sqrt{\sigma^2 + \epsilon}}$$

$$y = \gamma\hat{x_i} + \beta$$

where γ and β are parameters that the model learns.

- Masked Self-Attention Layer: attention is a mechanism that allows neural networks to assign a different amount of weight to each token in a sequence and process each token as a weighted average of all other tokens.
  In practice three matrixes are calculated:

- Q (Query): the representation of the current token
- K (Key): the representation of all the other tokens, which are matched with the current token
- V (Value): the representation of all the words, used for the weighted-average

The Q, K and V matrixes are initialized as:

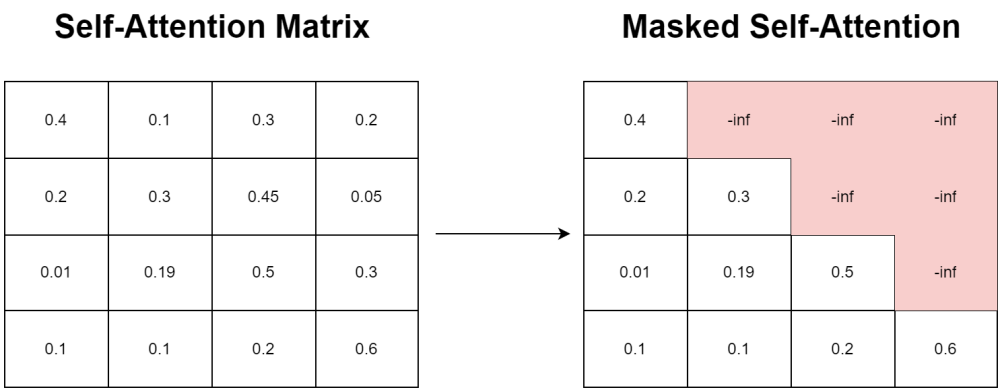
- $Q = W_q X + b_q$
- $K = W_k X + b_k$
- $V = W_v X + b_v$

where X is the input matrix and the other matrixes are randomly initialized and learned by the model. Finally, the attention score is calculated with

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where d is a normalization factor equivalent to the embeddings dimension The masked self-attention is a modified version of self-attention where all the tokens that appear after the current one are set to 0, in order not to let the model being influenced by any information regarding the tokens at the next positions. This is fundemental when training generative models such as GPT, whose scope is to predict the successive tokens.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T + Mask}{\sqrt{d}}\right)V$$



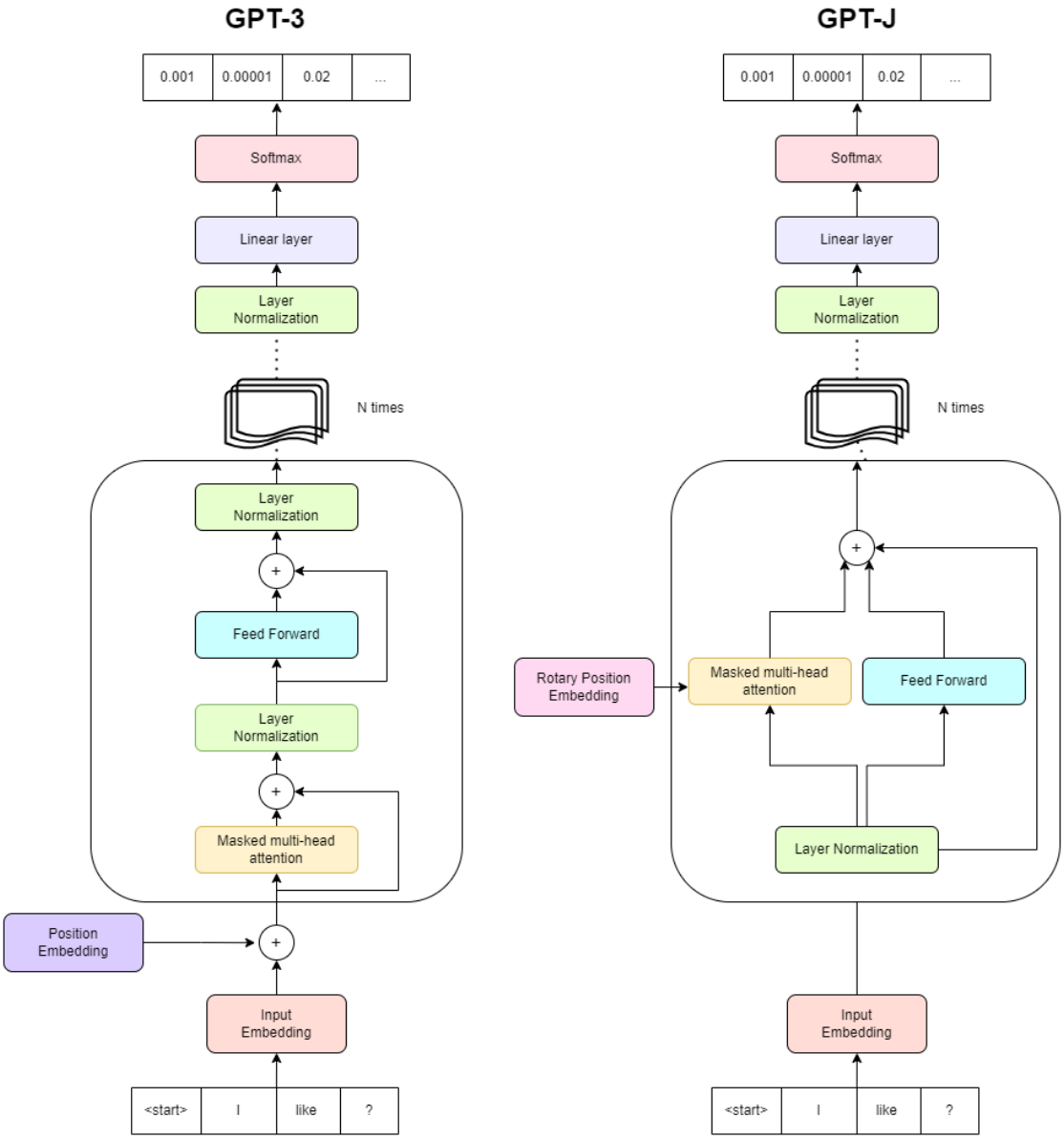


Figure 3.9: GPT-3 and GPT-J architectures compared

- Feed Forward Neural Network Layer: Used to add non-linearity to the transformer block

GPT models use a Byte-Pair Encoding tokenization. The Byte-Pair Encoding algorithm starts by building a vocabulary with all the single characters of the corpus we are training on. Then, at each step of the

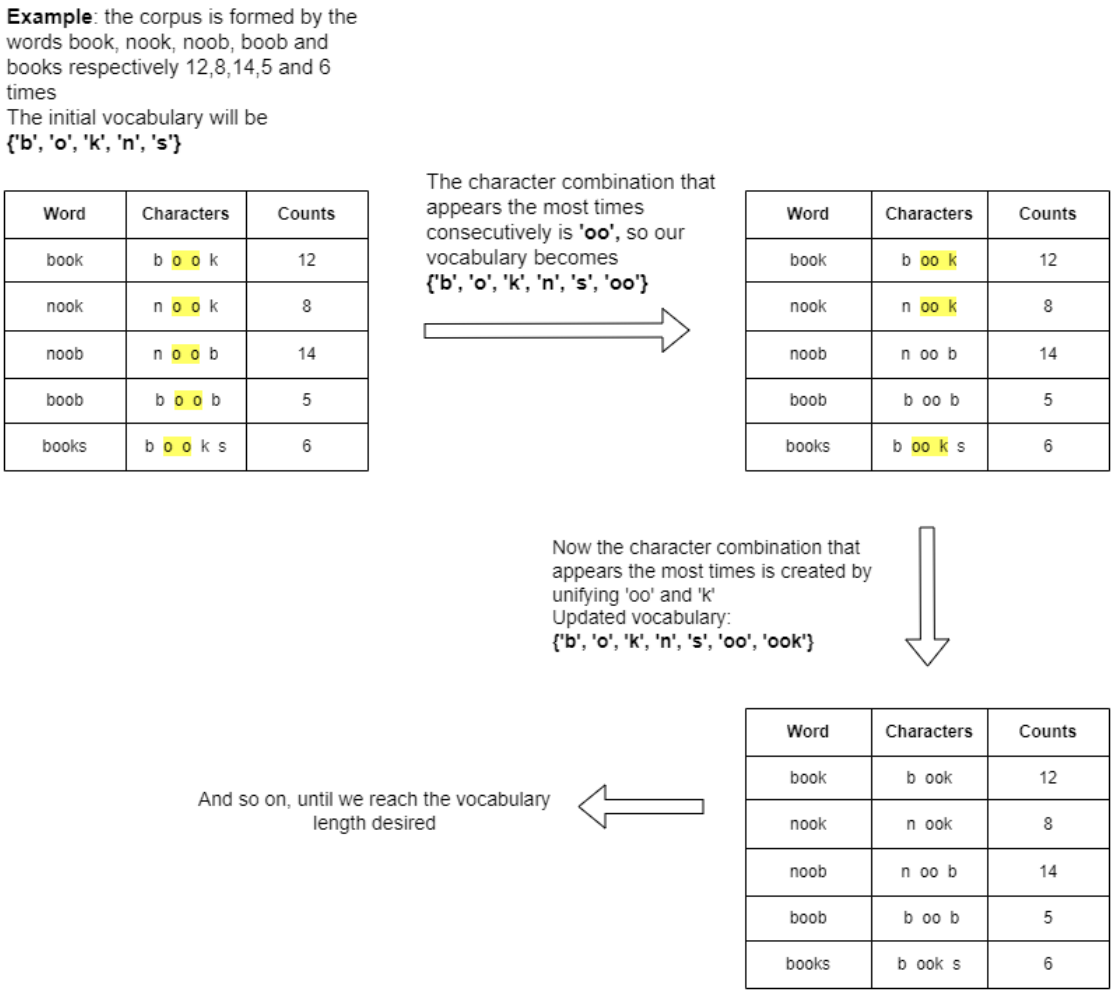


Figure 3.10: Byte-Pair Encoding Tokenization Example

algorithm, the two tokens which appear consecutevily the most in the words of the corpus are unified and create a new token. This process continue until the desired vocabulary lenght is satisfied. An example taken from the site [10] is shown in the Figure 3.10

Compared to GPT-3, GPT-J[11] has two minor architectural differences (shown in Figure 3.9):

- Rotary Embedding
- The attention layer and the feedforward layer in parallel for decreased

communication

Rotary Position Embedding

Position embeddings are used to infer the notion of position to the model, which does not have any sense of position for each token. In other words, using the attention mechanism each token "match" with the other tokens in the same manner, not considering where if the other token is located right after the current one or if it is at the end of the sentence. Position embeddings are used to add to the model this sense of position.
Rotary position embedding has been introduced by Su et al., it is a novel method that unifies absolute and relative approaches to position embeddings. The typical approach, that is also used by GPT-3, is to use a sinusoidal embedding, which is defined as

$$\begin{cases} p_{1,2t} = sin(k/10000^{2t/d}) \\ p_{1,2t+1} = cos(k/10000^{2t/d}) \end{cases}$$

where $p_{1,2t}$ is the 2^{th} element of the d-dimensional vector p_i RoPE instead proposes to incorporate the relative position information by multiplying the context representation with the sinusoidal functions instead of directly adding them.
If we define

$$\begin{aligned} q_m &= f_q(x_m, m) \\ k_n &= f_k(x_n, n) \end{aligned}$$

where f_q and f_k are functions that incorporates the m^{th} and n^{th} positions respectively to the vector embeddings x_m and x_n to produce the query and key vectors, we can require the inner product of the query q_m and k_n to be formulated by a function g that depends only on the word embeddings x_m, x_n and their relative position $m - n$

$$\langle f_q(x_m, m)\ ,\ f_k(x_n, n)\rangle = g(x_m, x_n, m - n)$$

In the simplest case $d = 2$, $f_{\{q,k\}}$ are defined as

$$f_{(\{q,k\},\{m,n\})} = (W_{\{q,k\}} x_{\{m,n\}}) e^{i\{m,n\}\theta}$$

and therefore we obtain

$$g(x_m, x_n, m-n) = Re[(W_q x_m)(W_k x_n)^T e^{i(m-n)\theta}]$$

which preserves the relative positional information of the word embeddings. This equation is used when calculating the self-attention, which will become

$$Attention(Q, K, V) = softmax\left(\frac{g(x_m, x_n, m-n)}{\sqrt{d}}\right)V$$

This equation can be generalized for $d > 2$, as shown in the original paper. In the end, incorporating the RoTE is pretty straightforward, you just have to rotate the word embedding by a multiple of its position index.
According to the researchers that published GPT-J[12], using RoTE leads to a faster convergence of training and validation losses and a lower overall validation loss.

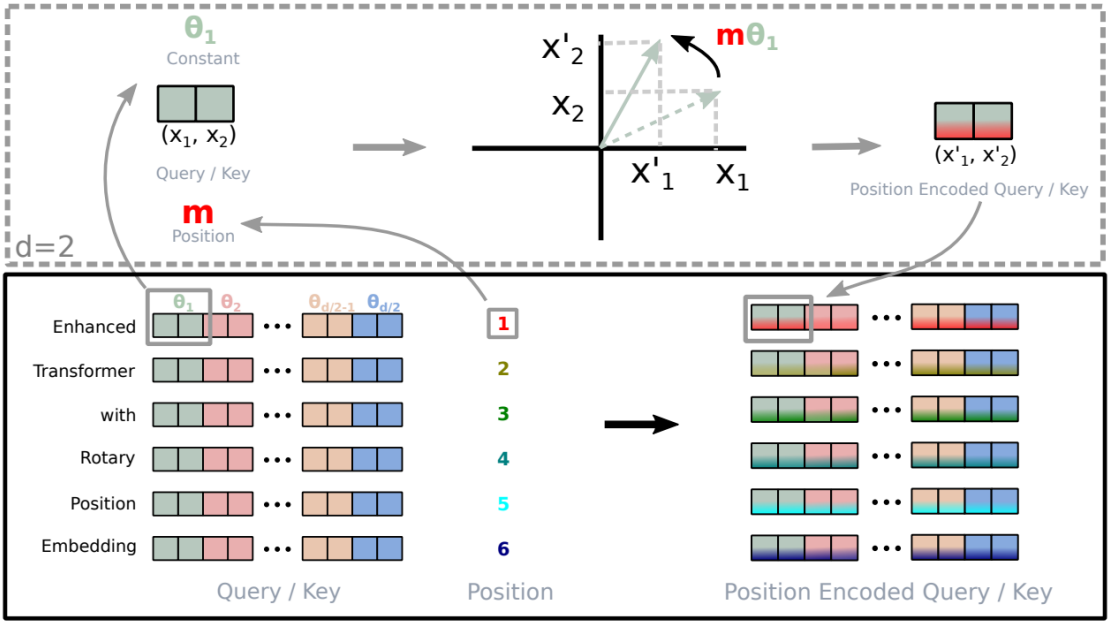


Figure 3.11: Implementation of RoPE, Image taken from original paper

3.6 Survey

In order to understand whether the tickets were similar to real tickets, we conducted a survey internal to the company. We were not able to get real

data from the company's HR due to the sensitive nature of the data.
We asked all our collegues to fill a form in an Excel file, where the users were given twenty randomly prompts, similar to the ones that are given to the GPT model.
After giving a brief explanation of the project and what we were trying to achieve, we recommended the users not to use their personal information while writing the fake tickets, but to use the same style of text and the same vocabulary they would use in a real situation.
For each ticket, there were a number of information attached that appeared over all the tickets, which were name of the person, category of the ticket and sub category of the ticket, and some information that were specific to the ticket's category, for example for the category-sub_category 'Salary - Salary Raise' the additional information given was the new requested salary.
We asked the users not to use necessarly all the information that were present in the prompts, but only the ones that felt natural to use, and to slightly changing them if it made sense to them (Ex: Sick leave from Oct 20 2022 can become 'from this Thursday, the 20')
Each user was given twenty different prompt sampled randomly from a set of 800 total prompts, belonging to the same categories-sub_categories of our taxonomy. The prompts contained only the contextual information of the ticket and no ticket's text, so, unlike the prompts used for the GPT models, there were no initial text such as 'Dear Sir/Madame, I would like to . . . '
The respondant were also asked to tick the information that they used, and these information was supposed to used as testing in the NER use case. However, many people either did not tick any information or they ticked only partially what thay used, therefore in the end we did not make use of these data.

3.7 Results

In this section we analyze the created dataset, comparing it to the tickets we collected from the survey.
The final dataset of HR tickets automatically generated is composed by exactly 16,000 tickets, 2000 for each category. On the other hand, we

Ticket details		Detail used (X if yes, empty if not)
First name	Zachary	
Last name	Traore	
Ticket category	Life event	
Ticket sub-category	Health issues	
Date start absence	18/07/2017	
Reason absence	COVID	X
Subject	Request for sick leave for 2 days	
	Ticket text	
	Dear Sir/Madame, I am positive to COVID and I cannot come to work for the next days. I don't exactly know when I will be able to return to work. Zachary	

Figure 3.12: Example of survey

collected 259 test tickets from 29 different respondants.
To evaluate the generation of the tickets, we have used a handful of traditional unsupervised text evaluation techniques. We were not able to exploit more complex classical metrics, such as BLEU and ROUGE, not having real data as references for each prompt. The reported metrics are:

- avg ttr unigram: average TTR(type token ratio) for the unigrams of the tickets

$$TTR = \frac{number\_of\_unique\_unigrams}{total\_number\_of\_unigrams}$$

- avg ttr bigram: average TTR(type token ratio) for the bigram of the tickets

$$TTR = \frac{number\_of\_unique\_bigrams}{total\_number\_of\_bigrams}$$

- avg ratios of nouns: average ratio of nouns in the tickets

$$noun\_ratio = \frac{counter\_noun\_words}{counter\_all\_words}$$

- avg ratios of verbs: average ratio of verbs in the tickets

$$verb\_ratio = \frac{counter\_verb\_words}{counter\_all\_words}$$

- avg_word_frequency: average frequency of words, using as frequency estimates a dump of the English version of Wikipedia, considering only words appearing at least 10 times

$$word\_freq = \begin{cases} log_e(word\_freq\_wiki), & \text{if } word\_freq\_wiki \geq 10 \\ skip, & \text{otherwise} \end{cases}$$

First, we reported the type-token-rations given that high-quality writing has been associated with the presence of more diverse words and phrases[13]. We computed them for each ticket and then we averaged the results.
Second, since lower frequency words indicate a more advanced output[14], we compute the average word frequency of the generated words.
Then, we calculate nouns and verb ratios over sentence length, as indicators of syntactic complexity, and therefore richer text[15]. To single out nouns and verbs, we uses the pre-trained Spacy POS model.

We report in Table 3.3 the results we obtained initially. The metrics have been calculated not only on the tickets generated by the model and the tickets of the surveys, but also on some baseline datasets (*Amazon reviews*, *Reddit comments* and *NIPS papers*). The datasets have been chosen to be as diversified as possible in terms of text longevity, type of language and terms used.
The main take aways from the are:

- The model tends to write more unique words and not to repeat itself, since the average TTR of both unigrams and bigrams are higher in the tickets of the survey

- The model tends to use more nouns than a real person

- The model tends the same number of verbs as a real person
- The average number of words (word count) varies a lot from category to category, both in generated and survey tickets

We know that these analysis are not completely reliable due to the low number of real data compared to the size of our dataset. However they are a good indication of how to improve the generation.
For this reason, we changed the parameters of the ticket creation and we recreated the dataset from scratch, in order to be more akin to the survey tickets.
The final results are shown in Table 3.4.
Some of the surveys' metrics change a bit compared to the first table, this is due to some late respondants to the survey, which skewed a little bit the final test metrics.

Topic analysis

Topic modeling is a type of statistical modeling technique used to discover the hidden topics and patterns in a corpus of text. It is a type of unsupervised machine learning method that automatically discovers topics from a collection of documents by analyzing words and phrases in each document. Topics are usually represented by word clusters or distributions that are related to each other. Topic modeling can be used to generate meaningful summaries about large collections of documents.
We used topic modeling to visualize in a summerized form the major topics of our dataset and to verify that the topics created correspond to the ones in the taxonomy.
The model of topic modeling we chose is BERTopic[16]. BERTopic generates document embedding with pre-trained transformer-based language models, clusters these embeddings, and finally, generates topic representations with the class-based TF-IDF procedure

In Figure 3.13 we show all the topics that BERTopic outputs, and in Figure 3.14 the grouped major 8 topics.
We are satisfied with the results, we can clearly recognize 7 of our 8 categories in the topics created, the only one missing is the request of time off due to

BERTopic

1: Documents are emdedded to create representations in vector space using Sentence-BERT
2: The embeddings generated are reduced in dimensionality using UMAP
3: The reduced embeddings are clustering used HDBSCAN. HDBSCAN is a variation of the hierarchical clustering algorithm DBSCAN that models clusters using a soft-clustering approach, allowing noise to be modeled as outliers
4: The most relevant term for each topic are found using a class-based version of TF-IDF. All documents in a cluster are treated as a single document by simply concatenating them, so we measure how much information a term provides to a class instead of a document.
5: If the user requests a number of topic smaller that the number of topics generated, the least common topics are iteratively merged with their most similar topic, until the number of topics request is satisfied.

Dataset	avg ttr unigram	avg ttr bigram	avg ratios of nouns	avg ratios of verbs	avg word frequency	avg word count
Amazon reviews	0.81	0.98	0.16	0.10	13.86, σ=0.47	73.73, σ=71.75
Reddit comments	0.91	0.99	0.15	0.11	13.33, σ=1.53	37.70, σ=84.96
NIPS papers	0.26	0.72	0.18	0.06	13.50, σ=0.25	5798.40, σ=737.17
Tickets survey	0.87	0.99	0.17	0.11	13.84	42.73, σ=28.83
Tickets generated by GPT	0.93	1.00	0.14	0.10	13.84, σ=0.35	57.87, σ=19.13
Survey Salary_Salary raise	0.85	0.99	0.20	0.10	13.85, σ=0.56	55.65, σ=35.67
GPT Salary_Salary raise	0.90	1.00	0.15	0.10	13.72, σ=0.33	70.73, σ=16.36
Survey Complaint_Complaint	0.83	0.98	0.15	0.13	13.66, σ=0.91	63.64, σ=46.67
GPT Complaint_Complaint	0.91	0.99	0.14	0.12	13.94, σ=0.32	72.81, σ=21.62
Survey Life event_Personal issues	0.89	0.99	0.17	0.12	13.99, σ=0.56	33.54, σ=19.85
GPT Life event_Personal issues	0.92	1.00	0.13	0.12	14.08, σ=0.25	63.88, σ=15.62
Survey Refund_Refund travel	0.90	0.99	0.17	0.11	13.56, σ=0.56	34.76, σ=17.02
GPT Refund_Refund travel	0.96	1.00	0.16	0.08	13.72, σ=0.38	38.91, σ=10.83
Survey Timetable change_Shift change	0.83	0.99	0.16	0.10	14.16, σ=0.66	38.15, σ=15.30
GPT Timetable change_Shift change	0.93	1.00	0.13	0.10	13.79, σ=0.37	47.95, σ=9.72
Survey Ask information_Accommodation	0.88	0.99	0.17	0.11	13.80, σ=0.50	39.93, σ=23.89
GPT Ask information_Accommodation	0.95	1.00	0.14	0.10	13.68, σ=0.36	47.10, σ=10.59
Survey Life event_Health issues	0.90	0.99	0.17	0.13	13.88, σ=0.44	31.92, σ=14.70
GPT Life event_Health issues	0.91	1.00	0.14	0.10	13.84, σ=0.33	59.07, σ=16.06
Survey Salary_Gender pay gap	0.85	0.97	0.21	0.11	13.87, σ=0.43	53.94, σ=32.71
GPT Salary_Gender pay gap	0.93	0.99	0.15	0.11	13.96, σ=0.28	62.52, σ=18.67

Table 3.3: Initial Results

personal reasons, that have been incorporated in Topic 3.
The

- Topic 0: Request of shift change
- Topic 1: Request of explanation of wage gap between genders

Dataset	avg ttr unigram	avg ttr bigram	avg ratios of nouns	avg ratios of verbs	avg word frequency	avg word count
Amazon reviews	0.81	0.98	0.16	0.10	13.86, σ=0.47	73.73, σ=71.75
Reddit comments	0.91	0.99	0.15	0.11	13.33, σ=1.53	37.70, σ=84.96
Tickets survey	0.86	0.99	0.17	0.11	13.89, σ=0.59	44.43, σ=27.46
Tickets generated by GPT	0.94	1.00	0.14	0.10	13.86, σ=0.39	49.22, σ=15.49
Survey Salary_Salary raise	0.84	0.99	0.21	0.10	13.89, σ=0.52	55.10, σ=32.44
GPT Salary_Salary raise	0.90	1.00	0.15	0.10	13.73, σ=0.35	62.87, σ=14.12
Survey Complaint_Complaint	0.82	0.98	0.15	0.13	13.74, σ=0.84	62.74, σ=42.83
GPT Complaint_Complaint	0.92	0.99	0.14	0.11	13.94, σ=0.33	59.46, σ=18.39
Survey Life event_Personal issues	0.88	0.99	0.17	0.12	14.03, σ=0.55	38.15, σ=20.86
GPT Life event_Personal issues	0.94	1.00	0.14	0.11	14.19, σ=0.30	45.49, σ=16.94
Survey Refund_Refund travel	0.88	0.99	0.17	0.11	13.61, σ=0.53	37.50, σ=17.36
GPT Refund_Refund travel	0.96	1.00	0.16	0.08	13.72, σ=0.38	39.52, σ=10.46
Survey Timetable change_Shift change	0.83	0.99	0.16	0.10	14.17, σ=0.62	38.73, σ=14.71
GPT Timetable change_Shift change	0.93	1.00	0.12	0.10	13.74, σ=0.40	42.80, σ=7.55
Survey Ask information_Accommodation	0.87	0.99	0.16	0.11	13.90, σ=0.50	41.86, σ=23.79
GPT Ask information_Accommodation	0.95	1.00	0.14	0.10	13.65, σ=0.38	46.79, σ=10.99
Survey Life event_Health issues	0.87	0.99	0.17	0.13	13.94, σ=0.42	37.91, σ=21.80
GPT Life event_Health issues	0.94	1.00	0.16	0.09	13.95, σ=0.37	43.36, σ=11.26
Survey Salary_Gender pay gap	0.85	0.97	0.22	0.11	13.87, σ=0.41	53.14, σ=31.13
GPT Salary_Gender pay gap	0.94	1.00	0.14	0.11	13.99, σ=0.29	53.49, σ=13.51

Table 3.4: Final Results

- Topic 2: Official complaint about collegue/superior
- Topic 3: Request of time off due to health reasons
- Topic 4: Request of salary increase
- Topic 5: Request of refund for work travel

- Topic 6: Request of refund for work travel

- Topic 7: Request of information for new accommodation

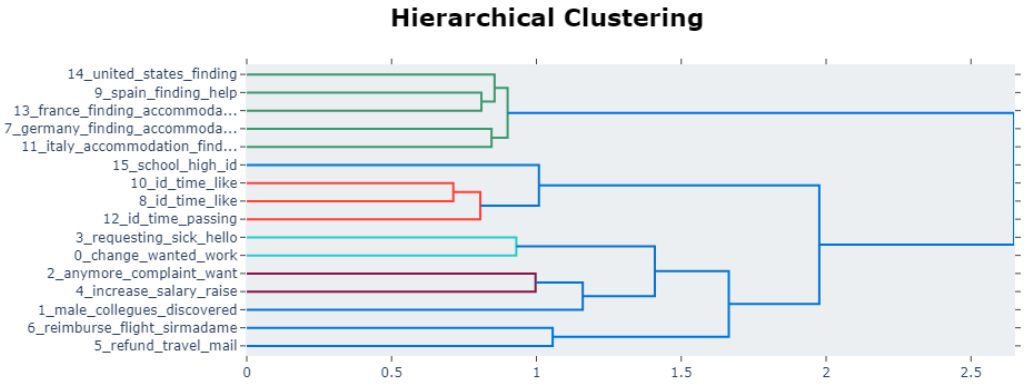


Figure 3.13: Hierarchy of all topics generated by BERTopic

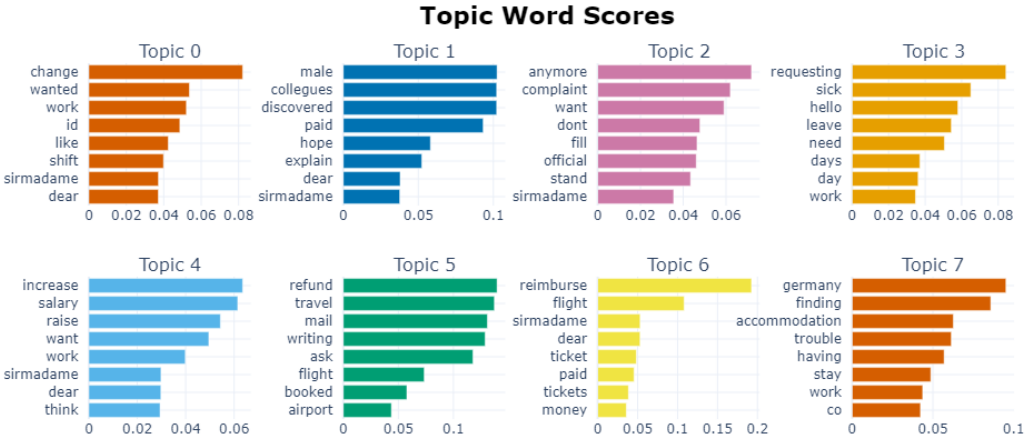


Figure 3.14: Main 8 topics generated by BERTopic

3.8 Additional Experiments

In this section, we are going to describe various methods we have tried for the generation of tickets which have not been included in the final version of the ticket generation for different reasons.
Some of these reasons are lack of computational power, not good enough results and lack of time.

3.8.1 Topical Generation

Zandie et al.[17] have presented topical language generation, which consists of generating text conditioned on a specific chosen topic. The topic can be chosen by the users from a predetermined list of topics. The topics are represented as a distribution over a vocabulary, and they are extracted from a corpus using algorithms of Topic Modeling.
The algorithm used is Latent Semantic Analysis, which gives us scores for each token per topic. LSA is able to discover latent topics in a document-term matrix by utilizing Singular Value Decomposition (SVD) to decompose the matrix.
The topics are inferred to the text generation by adding the embedding of the topic to the logits of the model before the last softmax layer, where the parameter γ control the topic inference (the higher the parameter γ is, the more the text generated will be about the topic chosen)

$P(t_j|\ x_i)$: probability of topic j given word k
$S(x_i|\ x_{<i})$: logits of token i

They used a threshold to increase the probability of only the tokens that already have a minimum probability to be sampled.

$$logprob(i) = \begin{cases} logP(t_j|\ x_i), & \text{if } S(x_i|\ x < i) \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

The final equation is:

$$P(x_i|\ x< i, t_j) = softmax(S(x_i|\ x_{<i}) + \gamma\ logprob(i))$$

We experimented this method to generate the tickets' texts. The topic-word matrix was calculated starting from 20 Newsgroups, a dataset of 18000 news posts on 20 topics.

Topic used: football

> [. . .]
> Ticket category: Life event
> Ticket sub-category: Health issues
> Date start absence: 24/08/2016
> Reason absence: a physiotherapy visit
> Subject: Request for sick leave for 1 day
>
> Dear Sir/Madame, my name is Adriana Giulietti. I am requesting football medical treatment on the following dates : - 28 nfl season 2016 (from 30th of August regular football game) professional training session in Milan from 29 football football league match between FC Portcullis football conference matches and other

Topic used: president, government

> [. . .]
> Ticket category: Life event
> Ticket sub-category: Health issues
> Date start absence: 25/12/2012
> Reason absence: Hordeolum
> Subject: Request for sick leave for 2 days
>
> Dear Sir/Madame, my name is Cristian Manso. I am requesting sick pay or vacation from you because state government has decided to cancel the contract which was signed on 20 th of December 2012 with head office called president and deputy leader states house republic functions as a general administration department in charge common services like public works system

The possible future implementations of the ticket generation could be completely executed with a unique prompt, letting the user choose the topic

from a set of pre-trained topics. This would be more flexible than the current method since to add new categories it would be sufficient to add a new row to the topic-word matrix.
However, the current results were not satisfying: the created tickets were not fluent and we did not find a corpus to create a topic-word matrix of topics related to the HR ticketing environment.

Chapter 4

Use Cases

In order to assess the suitability of the HR ticket dataset for downstream machine learning applications in HR, we exploit the obtained dataset using as labels the categories of the tickets.
The goal of testing different downstream machine learning tasks on the dataset is to show that our dataset is sufficiently rich to enable meaningful learning.
All the experiments have been carried out using as training data the dataset generated by the model, whereas the test data used was the tickets collected with the survey.
One of the main motivations to acquire the survey tickets was to have data that had no biases, or at least biases different from ours. Using a portion of the HR tickets dataset as test data would be meaningless, since the model would not necessarily be tested with data that was outside of the one it had been trained with. On the contrary, achieving good results on a portion of the HR ticket dataset would assert the performance of the models used and not the usefulness of the dataset.

4.1 Classification

Sentence classification is a natural language processing task that involves assigning a predefined category or label to a given sentence. One way to

approach sentence classification is to use word embeddings, which are numerical representations of words that capture their meaning and context within a sentence. Word embeddings can be generated using various techniques, such as training a neural network on a large dataset of text or using a pre-trained language model.
Once the word embeddings have been generated, you can input them into a classifier along with other features of the sentence, such as its grammatical structure, to make predictions about the sentence's category or label. The classifier uses the word embeddings as input to make predictions about the sentence's category or label.
We used two different pre-trained language model: BERT and fastText. We chose BERT because it is a widely used and well-studied model, and our objective was not to find the absolute best classifier, but only to show that the dataset could be used as training data.
Instead, we chose fastText to have an option which was CPU-friendly, and that could be trained in a matter of a few minutes.
The training data used were the texts of the 16000 tickets that compose the HR ticket dataset, whereas the test data were the survey tickets. The labels were the combination of ticket category and ticket subcategory of each ticket (Ex. "Life event_Health issues").

4.1.1 FastText Classifier

FastText is a library developed by Facebook AI Research that provides a set of training algorithms for supervised learning of word embeddings from raw text and also can be used to build and train supervised text classification models.
FastText creates word embeddings using a combination of character n-grams and word n-grams. To do this, fastText first breaks each word in the training text into its constituent character n-grams. For example, the word "cat" might be broken into the character n-grams "c", "ca", "cat", "a", "at", and "t". These character n-grams are then used to generate vectors. The word embeddings are calculated as the sum of their n-gram vectors.
To learn the embeddings of these n-grams, fastText uses a skip-gram model, which is a type of model that maximizes the probability of observing a word given its context, or in other words its surrounding words. By using

this approach, fastText is able to learn high-quality word embeddings that capture the syntactic and semantic information of words.
The fastText classification uses the fastText embeddings as input to the model, and in particular:

fastText Classification
1: Words representation are averaged into a text representation
2: The text representation is fed to a linear classifier
3: The output of the linear classifier is given as input to a softmax function to compute the probability distribution over a set of predefined classes

4.1.2 BERT classifier

BERT (Bidirectional Encoder Representation from Transformers) is a model developed by Google AI based on the transformers architecture. BERT was trained on a large corpus on two different tasks: Masked Language Modeling and Next Sentence Prediction.
Through the use of MLM, in the training phase, 15% of the tokens in a sentence are obscured and BERT can then be utilized to utilize the surrounding words in both directions to predict the masked tokens, facilitating bidirectional learning from the text.
On the other hand, NSP helps the model understand the relationships between sentences. Specifically, in the training phase, 50% of the examples are actually successive sentences, while in the other 50% of the cases this condition is not respected.
BERT is quite straightforward to fine-tune, we just need to plug in the necessary layers at the top of the BERT architecture and fine-tune for a sufficient number of epochs the model end-to-end.
The first token of a BERT embedding is the [CLS] token, which is a special token that acts as an aggregate representation of the sentence. The final embedding of the [CLS] token is used as input for the classification task. The most simple method, which is also the method used by the BERTForSequenceClassification class by HuggingFace, is to take the embedding of the [CLS] token at the last hidden layer and fed it to a single layer of a feed-forward network. The layer of the feed-forward network will have n

input, where n is the dimension of the embedding of a token, and m output, where m is the desired number of outputs.
The final model is shown in Figure 4.1

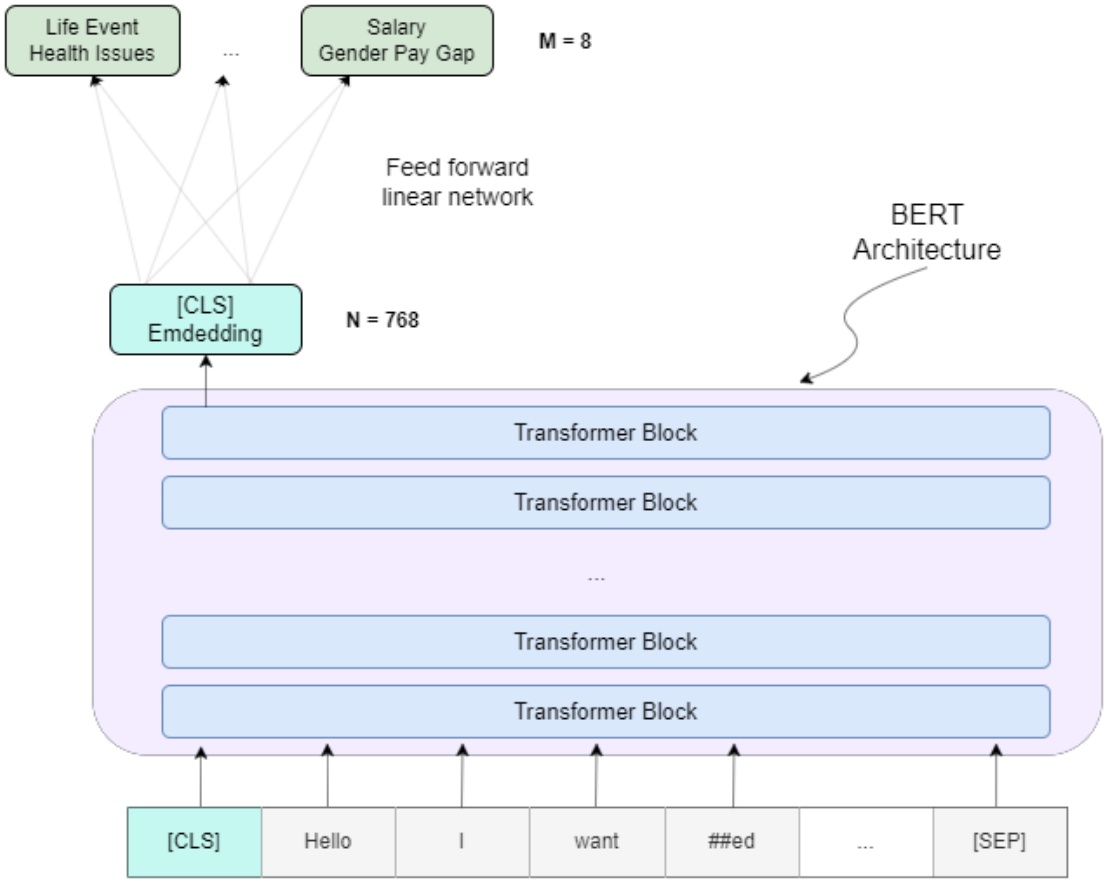


Figure 4.1: Schema of BERT classification

4.1.3 Results

In this section we show the results of the classification experiments. We want to reiterate that the point of the experiments was not to get the best possible results, therefore the models were not trained for a large amount of epochs and we did not try an excessive number of possible hyperparameters. The hyperparameters tried are shown in Table 4.1 and Table 4.2. If an hyperparameter is not shown in the table it means we have used its default

value.
The technique used for finding the optimal hyperparameters is Grid Search, which means we have trained and tested the models for each combination of all the chosen values of the hyperparameters to determine the hyperparameters' set which achieves the best results.

Size of context window (ws)	5
epochs	20
minimal number of word occurencies (minCount)	1
max length of word ngram (wordNgrams)	3
learning rate (lr)	0.5
learning update rate (lrUpdateRate)	100
sampling threshold (t)	0.0001

Table 4.1: Hyperparameters fastText

epochs	[3, 5]
learning rate	[1e-05, 5e-04, 1e-04, 5e-05, 5e-06]
weight decay	[0.001, 0.01, 0.05]
train batch size	[8, 16]
warmup steps	500

Table 4.2: Hyperparameters BERT

As we expected, fastText performs worse than BERT, achieving an f_1 score of only 0.41.
On the other hand, the BERT classifier with hyperparameters

- epochs: 5
- learning rate: 5e-05
- weight decay: 0.001
- train batch size: 8
- warmup steps: 500

achieved an f_1 score of 0.78. We show also the confusion matrix(Figure 4.2) to highlight how the majority of errors comes from two sub-categories that

belong to the same category, in particular "Gender wage gap"-"Salary raise" and "Personal issues"-"Health issues". In some way, this validates our initial decision on the taxonomy of the tickets.

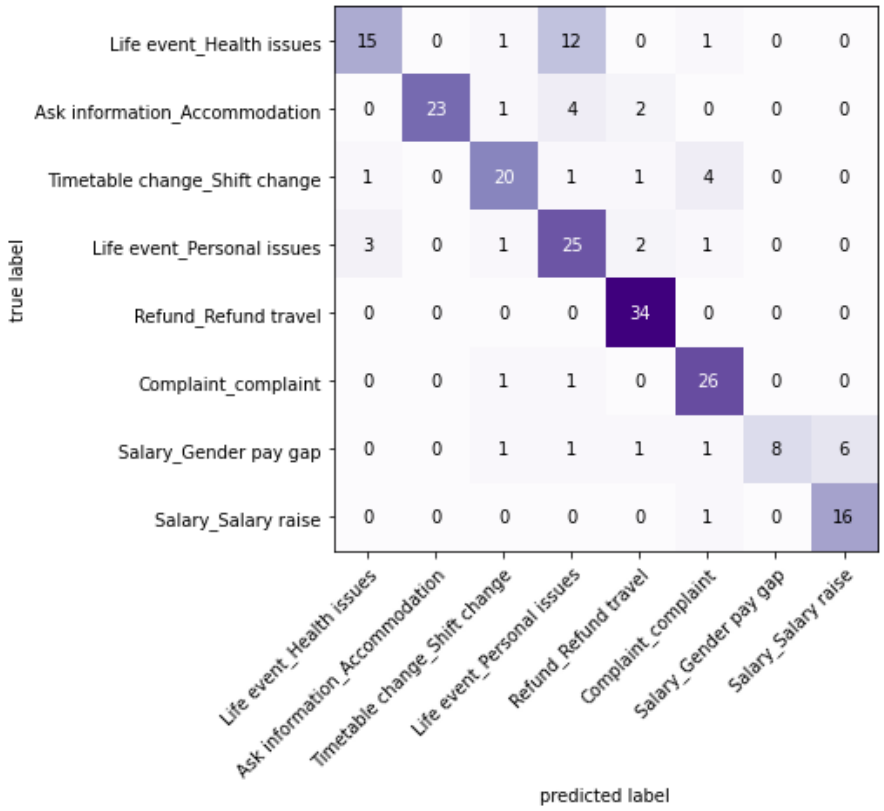


Figure 4.2: Confusion matrix of Ticket classification with BERT

4.2 Anonymization

Anonymization of personal data refers to the process of removing personally identifiable information (PII) from data sets so that the individuals represented in the data cannot be identified. This is accomplished by either completely removing or replacing identifiable data with generic values. The purpose of anonymization is to protect the privacy of individuals while still

allowing the data to be used for legitimate purposes.
Most of the recent implementations of anonymization work by masking the personal information of a person, such as names and surnames, telephone numbers, addresses, credit card numbers and so on.
One of the most famous libraries for anonymization is Presidio, developed by Microsoft. Presidio exploits pattern recognition with regex and Named Entity Recognition to find all the personal information and mask them.
The main disadvantage of such techniques is that often the personal subject can be identified through the so-called quasi-identifiers, that more often than not are not masked.
Here reported some examples that are not masked by Presidio:

Original sentence

> The new intern at my office, the one with red hair, caught covid last week

Sentence redacted by Presidio

> The new intern at my office, the one with red hair, caught covid <DATE_TIME >

Original sentence

> The boss of the HR department has made some weird comments about how I dress

Sentence redacted by Presidio

> The boss of the HR department has made some weird comments about how I dress

Our new approach exploits a Sequence to Sequence model called T5 T5 is a model released by Google in 2019, it is a standard encoder-decoder transformer that, unlike BERT, always returns strings as outputs. This is why it is called a sequence-to-sequence model.
T5 is a unified model that can be applied for many downstream tasks, such as sentiment analysis, sentence completation, question answering...
In T5 architecture there are *adapted layers* after each feed-forward layer,

whose scope is to diminish the number of parameter updates for each fine-tuned model. In fact, *adapted layers* are dense-ReLU-dense blocks that are designed so that their input dimensionality and output dimensionality are equal. This lets us insert them into the Transformer architecture without other changes. When fine-tuning for each different task, only the *adapted layers* and the normalization layers will be updated. This results in a considerable reduction of parameter updates.
We followed a few-shot learning approach with T5. Rather than relying on a large amount of training data to allow a pre-trained model to adjust to a given task accurately, few-shot learning employs the use of a few examples to direct the training of a machine learning model with very minimal data. For each category of tickets, we wrote 10 examples of "anonymized" tickets, where we removed all personal information and information that could be retraced to the original writer, maintaining only information that could be useful for analysis purposes. Here are a couple of examples:

Original sentence:

> Dear Sir/Madame, I cannot stand anymore this discrimination and prejudice against French people at work

Anonymization used for few-shot learning:

> The employee is filling a complaint for discrimination based on nationality

Original sentence:

> Hello, my name is Zacaredas Pinilla and I work at Laguna-Franco Spain. I am having trouble finding accommodation in the Algeciras area so if you could help me with this matter it would be greatly appreciated

Anonymization used for few-shot learning:

> The employee is asking for help finding an accommodation

To evaluate the goodness of anonymization and how much of the information has remained after the anonymization we have used three different methods:

- **QAGS**: Questions Answering and Generation for summarization

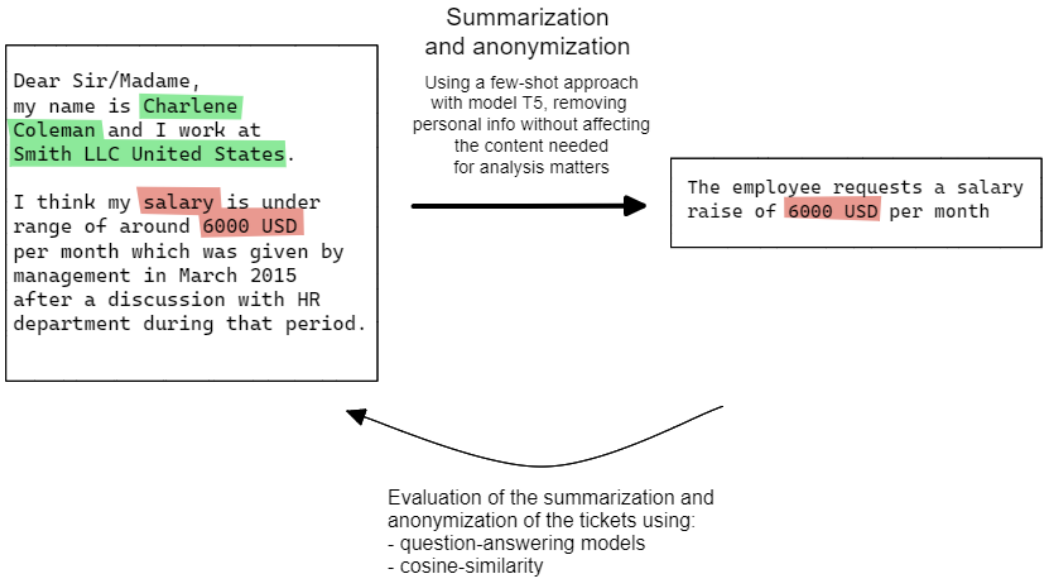


Figure 4.3: Schema of Ticket Anonymization

- **SUMMAQA**: Unsupervised metric for reinforced summarization model
- **Cosine Similarity**

These models were created originally to evaluate the goodness of a summarization, we have adapted them to be used for evaluating our anonymization. The first two methods follow the same philosophy: they generate questions to ask both the original version and the anonymized version of the tickets, looking if the answers are coherent. We call them Questions and Answers models. However they have few key differences. SummaQA do not create natural sentence questions but uses as questions the masked version of the sentences. Instead QAGS create natural language questions using a pre-trained model.

QAGS

QAGS is an automatic evaluation protocol that is based on the assumption that if we reply to questions considering as factual bases the summary and the source, we can determine if the summary/anonymization is consistent

with its source by comparing the answers.
QAGS works like this:

QAGS
1: A BART model is fine-tuned for question generation: the model receives both the answers and the source article from the NewsQA dataset and is trained to maximize the likelihood of the paired question
2: At test time, named entities and noun phrases are extracted from the context using spaCy and are considered as answers candidates. The summary is used as the context.
3: The BART model generates question based on the summary and its entities
4: A BERT model is fine-tuned for Question Answering on the SQuAD dataset
5: We answer with the help of the fine-tuned BERT model the questions generated beforehand using both the source article and the summary to get two sets of answers.
6: We compare the corresponding answers using an answer similarity metric, and we get a final score averaging the answer similarity metric (f1 score) over all questions

SUMMAQA

SUMMAQA works similarly to QAGS, the main difference is that there is not a generative model to create the questions, in particular:

Cosine Similarity

We measured the cosine similarity between sentence embeddings of original text and summarization. The sentence embeddings used were calculated using Sentence-BERT, a modified version of the pretrained BERT network that uses siamese networks to obtain more meaningful embedding of the sentence, compared to getting the embedding of the [CLS] token or to average all the other emeddings.

SUMMAQA

1: We find all the entities in the source text (the original ticket) with Spacy
2: Create one sentence (which will be our question) for each entity. We keep only the sentence in which the entity is if there are more sentences. The entity will be masked. Ex: "Yesterday I went to Paris" becomes "Yesterday I went to [MASK]"
3: The MASKED entities will be the True labels when calculating the metrics
4: We answer to the questions generated before using a fine-tuned version of BERT model, using both the source and the summarization/anonymization as contexts
5: We compare the answers and calculate the f1 score for both contexts

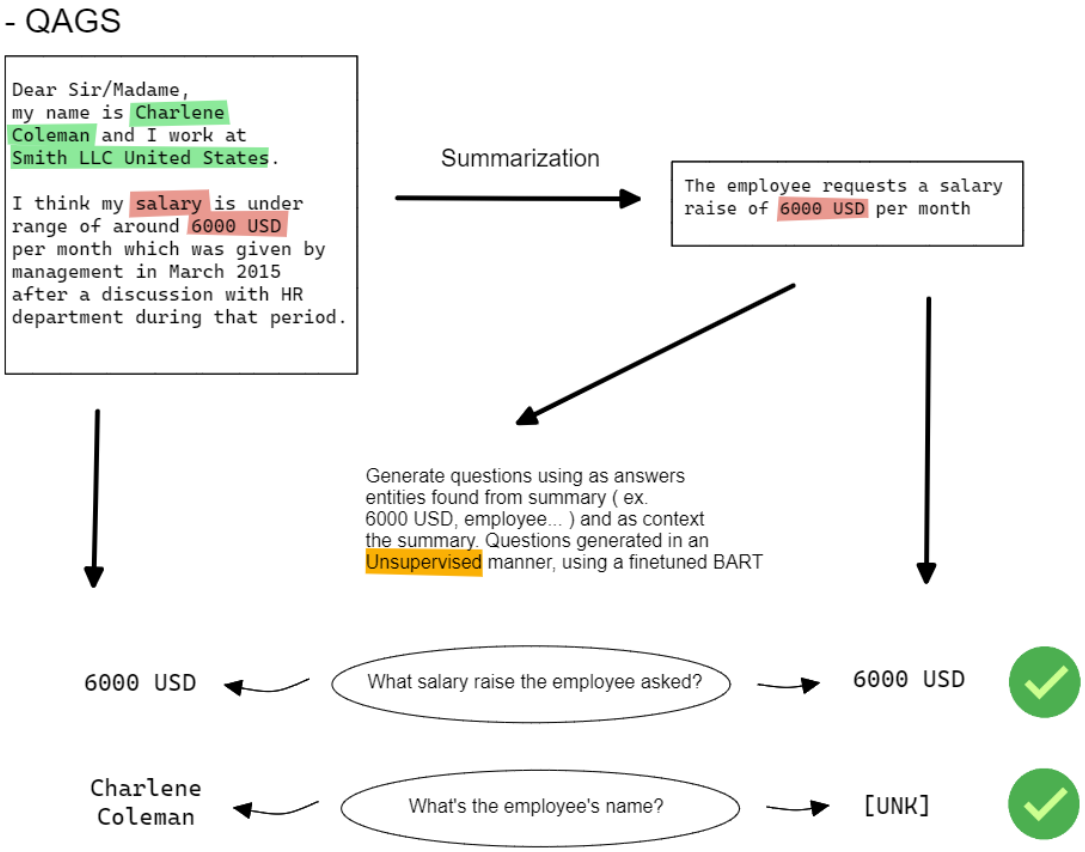


Figure 4.4: Schema of QAGS

4.3 Named Entity Recognition

Named entity recognition is the process of automatically identifying and classifying named entities in a text. This can include identifying and categorizing named entities such as people, organizations, locations, and so on. NER systems might be used to automatically extract information from a large collection of documents, such as identifying all mentions of specific named entities or analyzing the relationships between different named entities.

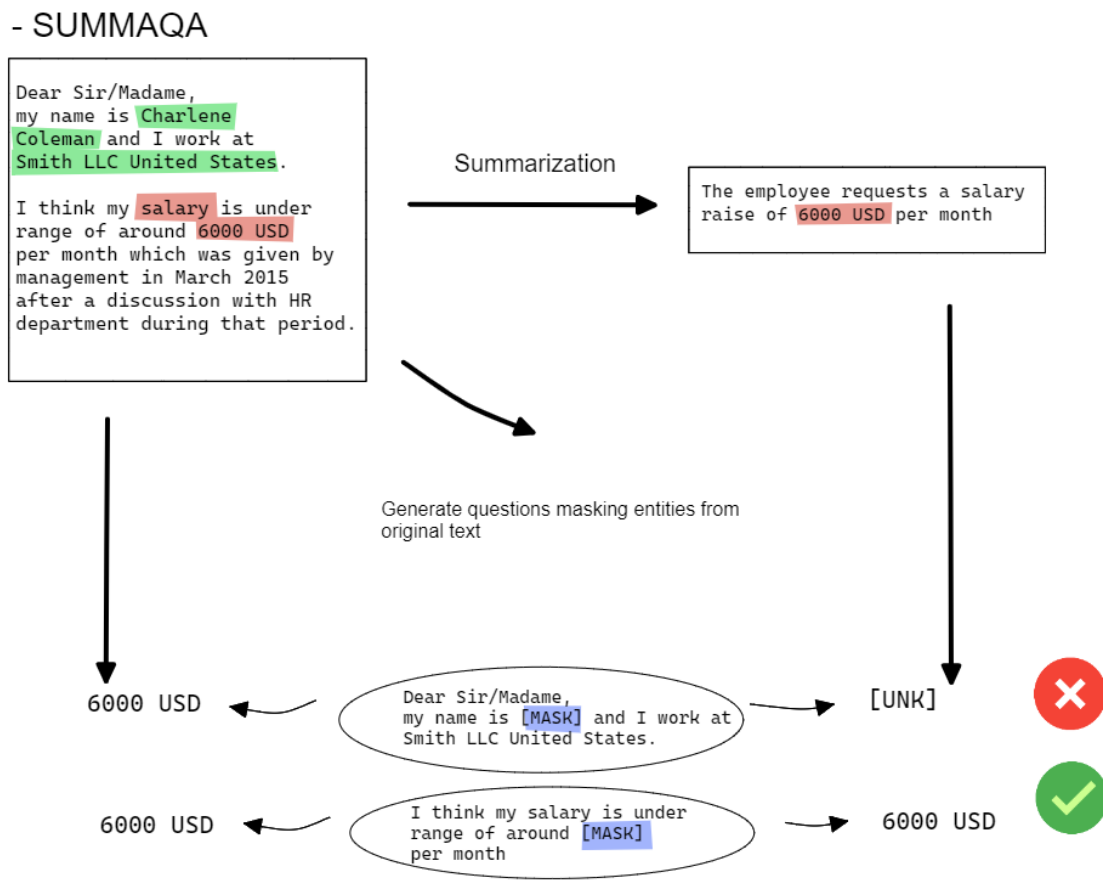


Figure 4.5: Schema of SUMMAQA

4.3.1 Weak labeling

During the creation of the ticket, we tried to automatically save also the entities of the ticket. With entities, we do not mean the classical entities used for NER, but the original variables specific to a ticket's category (complete list in Table 3.1).
Finding the entities in the prompt is trivial, since the positions are fixed in the template, and the filling operation is managed by us.
Example:

From: ${email}
To: ${company email}
First name: ${first name}

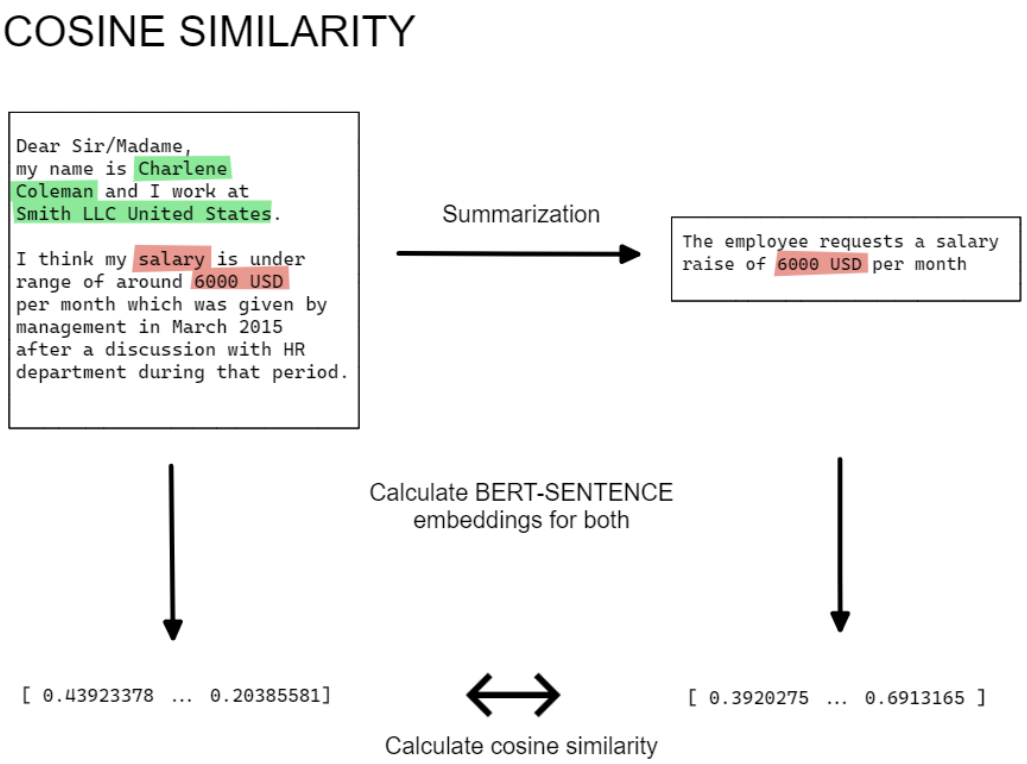


Figure 4.6: Schema of Cosine Similarity for

Last name: ${last name}
Company: ${company}
Date: ${ticket date} Date start absence: ${date_start_absence}
Reason absence: ${reason_absence}
Subject: Request for sick leave for ${days}

Though the template and prompt serve as our starting point to generate the actual ticket text, it is only this latter part that would be present in a real ticket. Identifying the locations of the entities is not straightforward since I usually provide only the initial information and a limited template, with the rest of the generation being done by GPT.
Once the ticket is generated, to find the entities in the generated text I use 2 different approaches:

- Exact match
- Heuristics

The exact match is straight-forward, it just looks for the exact same words in the generated text, and if the word/words are found, the entity is added to the list of entities of the ticket However this method often does not work, since GPT could use only part of the entity (Ex: "A medical consultation" is in the text as "a consultation") or slightly alter the entity (ex: 01/10/1998 → the first of October of 1998)
This is the reason why we implemented manually a set of different rules to find modified versions of the original entity in the text. These versions do not cover every possible case, since they are hard-coded by us from experience and by looking at how GPT behaves, this is why we called this approach "Heuristics".
Some of these heuristics are:

- Check not only the full entity, but also subsets of it removing stopwords
  Example:
  reason_of_absence: "An Urinary Tract Infection" →
  - "Urinary Tract Infection"
  - "Urinary Tract"
  - "Tract Infection"
  - "Urinary"
  - "Tract"
  - "Infection"
- Check all the possible version of a percentage text
  "5%", "5 %", "5.0%", "5 percent", ...
- Check different formats of date
  MM/DD/YYYY, DD/MM/YYYY, "First of October", "1st of October", ...

With this method, we were able to identify entities in only 6272 tickets out of the total 16000 tickets. The complete list of all the entities found is shown in Table 4.3.

duration	1016	increase_in_percentage	539
location	1411	work_title	33
date	70	wage_gap	421
work_shift	70	number_of_days	982
reason_of_change	7	date_start_absence	3
to_who	1221	description_life_event	1214
reason	1119	date_travel	112
complaint	106	airport	28
salary	376		

Table 4.3: Entities found with heuristics

4.3.2 Classical NER

Token classification is a common technique used in NER to identify and classify named entities in a text. In token classification, the text is first divided into individual tokens, which are typically words or phrases. The model then predicts a label for each token, indicating the type of named entity it represents.
To assign the labels to the tokens we used the BIO(Beginning, Inside, Outside) format, where the B tag is used for the first token of an entity, the I tag for the rest of the tokens of the entity and lastly the O tag is used for the token that do not belong to any entities.
Example:

Alex B-PER
is O
going O
to O
Los B-LOC
Angeles I-LOC
in O
California B-LOC

The label assigned to each token are then used for a token classification task, using a RoBERTa model and the Spacy training pipeline.
The used the default Spacy parameters for the training.

The survey tickets, used as always as our test set, were manually labelled by ourselves.
The results were not satisfying, we achieved a f_1 score of 0.34 on the exact matching and a f_1 score of 0.45 on the partial matching. When the model finds the correct entity for all the tokens and only the tokens of an entity, it's considered an exact matching. Whereas, wehen the model finds the correct entity for only a subset of the token of an entity, then is considered a partial match
Example exact match:

to O
Los LOC
Angeles LOC
in O

Example partial match:

to O
Los LOC
Angeles O
in O

The main problem we encountered was that the entities recognition was not able to recognize the difference entities of the same type depending on the context. For example, the model could not recognize the difference between a date of start absence (entity of request of time off due to health reasons) and a date of travel (entity of request of refund for travel).

4.3.3 Our approach to NER

To overcome the limits of the first approach, we developed a new approach based on transfer learning and sentence classification.
The main intuition behind the new approach is to use the pre-trained spacy NER model and exploit the entities found by the base model to extract our entities.

First of all, we build a dictionary of the matchings between entities of spacy and our entities. We decided to work on a subset of our entities and not consider all the entities that did not have a clear matching with Spacy entities. In Table 4.4 we show all the matchings. Then, we scan our dataset and we

Category	Entities matching
Ask information_Accommodation	{"location": "GPE", "duration": "DATE"}
Life event_Health issues	{"date_start_absence": "DATE", "number_of_days": "DATE"}
Refund_Refund travel	{"date_travel": "DATE", "location": "GPE"}
Salary_Gender pay gap	{"wage_gap": "PERCENT"}
Salary_Salary raise	{"increase_in_percentage": "PERCENT", "salary": "MONEY"}
Timetable change_Shift change	{"date": "DATE"}

Table 4.4: Matching of our entities with Spacy entities

look for each categorys the entities shown in Table 4.4. If we find an entity not present in the table, it is filtered out. For each entity we will have an array of the type [ENTITY, START_CHAR, END_CHAR] (Figure 4.7).
After that we tokenize the text of all the tickets and we look up for each ticket its entities tokens. Then for each entity we save the indexes of the corresponding tokens, so for example if the entity "twelve month" is split into two tokens "twelve" and "month", which are the 22^{nd} and 23^{rd} tokens of the text, then we will save ["twelve month", DATE, [22,23]].
We also had to manage the case in which a token is not present in the tokenizer vocabulary, so is splitted into sub-tokens. We took advantage of the fact that if a word is splitted in two or more words the new tokens will have the prefix "##".
The number of tokens for each entity can vary, for practical reasons in the training phase we set the maximum number of tokens to $N = 50$ (Figure 4.8). Now we have all we need for the training. The base architecture is a *bert-base-cased* model, which we modified on the last layer.

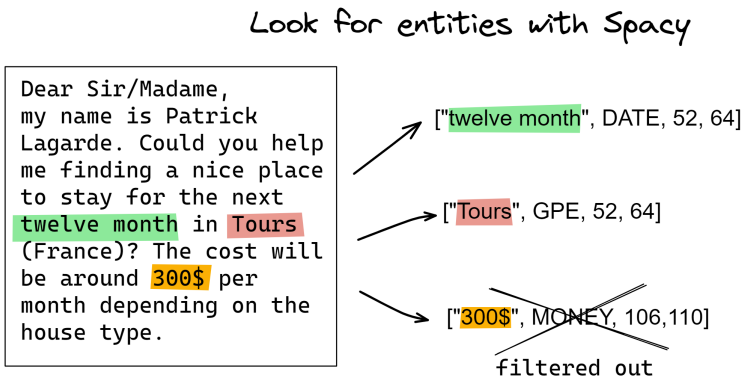


Figure 4.7: Ticket NER preprocessing part 1

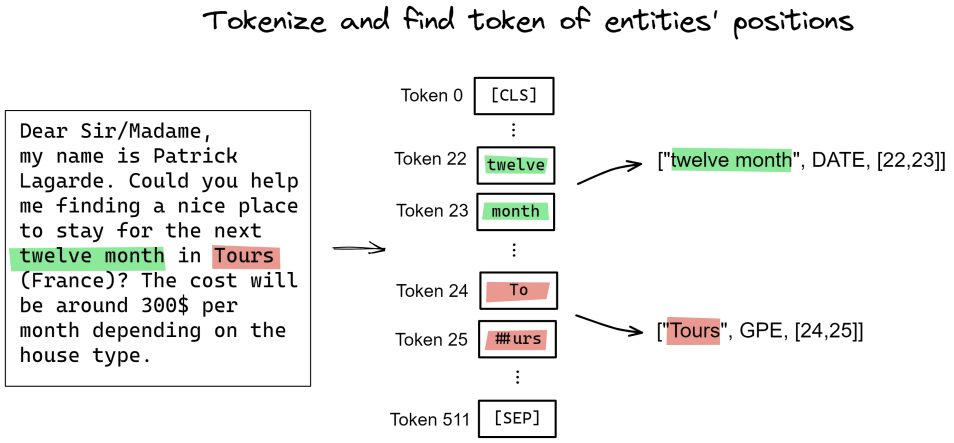


Figure 4.8: Ticket NER preprocessing part 2

In the training phase, we pass to the BERT model at each step the tickets' texts and their entities with the indexes of the matching tokens. For each entity of the ticket there will be a training record. The ticket will be unpacked into 512 tokens, and each token will be processed by the BERT architecture (same architecture as the classifier Figure 4.1).
At the last layer, instead of passing the [CLS] token to a classifier, we concatenate the [CLS] token with the average of all the embeddings of the tokens of the current entity. This is the reason why we preprocessed the dataset to find the indexes of the tokens' indexes for the entities. In BERT base each token has a dimension $d = 768$, so the input of the last feed forward

neural network will be $N = 1536$. The output dimension will be $M = 10$ (Figure 4.9). The labels of the training are the combination of the ticket category and the entity, since for each entity found in a ticket we generate a record (For example if there are 3 entities in a ticket, there will be 3 records in the training set with the same ticket text, but that will have a different token embedding for the classification in the last layer and they will have different labels)

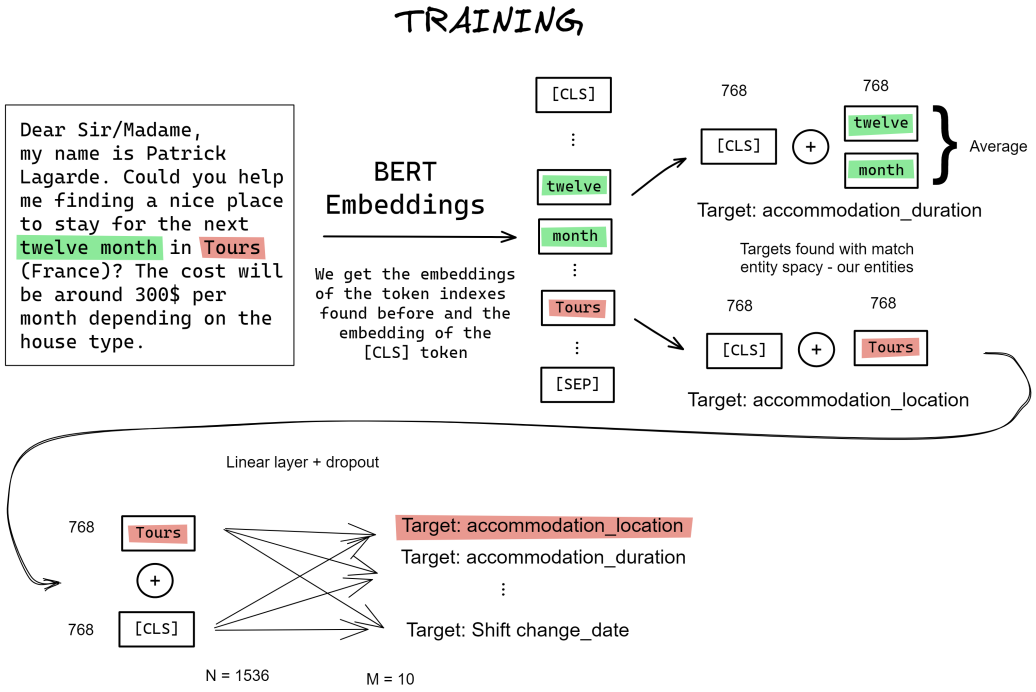


Figure 4.9: Ticket NER training

Bibliography

[1] SAP. *SAP Security Research*. `https://www.sap.com/documents/2022/03/8c5580cc-207e-0010-bca6-c68f7e60039b.html`. [Online; accessed]. 2022.

[2] Steven M Bellovin, Preetam K Dutta, and Nathan Reitinger. «Privacy and synthetic datasets». In: *Stan. Tech. L. Rev.* 22 (2019), p. 1.

[3] Ildikó Pilán, Pierre Lison, Lilja Øvrelid, Anthi Papadopoulou, David Sánchez, and Montserrat Batet. «The text anonymization benchmark (tab): A dedicated corpus and evaluation framework for text anonymization». In: *arXiv preprint arXiv:2202.00443* (2022).

[4] Yi-Chun Chen, Tim A Wheeler, and Mykel J Kochenderfer. «Learning discrete Bayesian networks from continuous data». In: *Journal of Artificial Intelligence Research* 59 (2017), pp. 103–132.

[5] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. «Language models are unsupervised multitask learners». In: *OpenAI blog* 1.8 (2019), p. 9.

[6] Tom Brown et al. «Language models are few-shot learners». In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[7] Leo Gao et al. «The pile: An 800gb dataset of diverse text for language modeling». In: *arXiv preprint arXiv:2101.00027* (2020).

[8] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. «Layer normalization». In: *arXiv preprint arXiv:1607.06450* (2016).

[9] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. «Understanding and improving layer normalization». In: *Advances in Neural Information Processing Systems* 32 (2019).

[10] LucyTalks. *The Modern Tokenization Stack for NLP: Byte Pair Encoding*. `https://lucytalksdata.com/the-modern-tokenization-stack-for-nlp-byte-pair-encoding`. [Online; accessed]. 2022.

[11] Ben Wang and Aran Komatsuzaki. *GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model*. `https://github.com/kingoflolz/mesh-transformer-jax`. May 2021.

[12] Stella Biderman, Sid Black, Charles Foster, Leo Gao, Eric Hallahan, Horace He, Ben Wang, and Phil Wang. *Rotary Embeddings: A Relative Revolution*. `blog.eleuther.ai/`. [Online; accessed]. 2021.

[13] Emily Pitler and Ani Nenkova. «Revisiting readability: A unified framework for predicting text quality». In: *Proceedings of the 2008 conference on empirical methods in natural language processing*. 2008, pp. 186–195.

[14] Scott A Crossley, Jennifer L Weston, Susan T McLain Sullivan, and Danielle S McNamara. «The development of writing proficiency as a function of grade level: A linguistic analysis». In: *Written Communication* 28.3 (2011), pp. 282–311.

[15] Danielle S McNamara, Scott A Crossley, and Philip M McCarthy. «Linguistic features of writing quality». In: *Written communication* 27.1 (2010), pp. 57–86.

[16] Maarten Grootendorst. «BERTopic: Neural topic modeling with a class-based TF-IDF procedure». In: *arXiv preprint arXiv:2203.05794* (2022).

[17] Rohola Zandie and Mohammad H Mahoor. «Topical language generation using transformers». In: *Natural Language Engineering* (2021), pp. 1–23.