Personal Data Detection in Free Text

Gabriele Gioetto

November 15, 2022

Contents

1 Introduction

An HR (Human Resource) department in a large organization receives inquiries/ requests from employees on multiple topics, 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 identifythe 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 that 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.

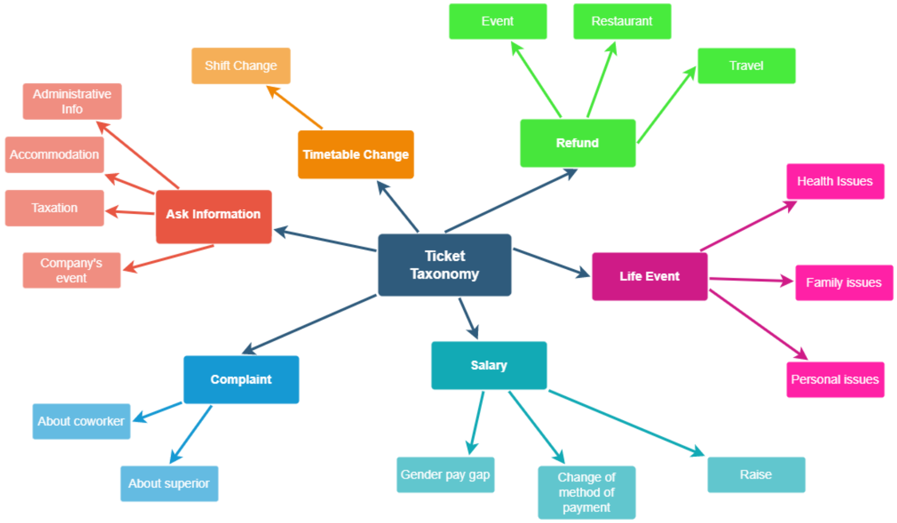
2 Related Work

3 Method

3.1 Taxonomy

Usually HR tickets can belong to various categories, which can range from a request of shift change to a complaint about a superior.
We built a taxonomy of tickets, which is structered in categories and sub categories. Each sub category has its own variables that are used as inputs for the ticket generation. For example to create a request of 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 has been built following the advise of an HR expert from SAP, however the final taxonomy

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 can be seen in the Table 1



Category	Sub-category	variables
Ask Information	Accomodation	location, duration
	Taxation	issue
Complaint	About coworker	complaint
	About superior	complaint
Timetable change	Shift change	reason_of_change, old_date, new_date
Salary	Salary raise	old_salary, new_salary, increase
	Gender pay gap	wage_gap
Life Event	Health issues	disease, number_of_days_of_sick_leave
	Personal issues	issue, number_of_days
	Family issues	member_of_family, issue, number_of_days
Refund	Event	date, type_of_event
	Travel	from, to, date, vehicle
	Restaurant	location, date

Table 1: Table of all defined categories and sub-categories

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 dataset 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 of the absence (almost always a disease) and the number of hours of absence. It is used to create the requests of 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 the requests of salary raise.
- *List of events of life*: list of major events in life. It is used to create the requests of time off due to personal reasons.
- *Gender pay gap in the UK*: dataset of employers with 250 or more employees, comparing men and women's average pay across the organizations. It is used to create the requests of explanation for the wage gap amongst genders.
- *OpenFlights database*: datasets of airports and flights all over the world. It is used for the requests of refund of travels.
- *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 the requests of 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 of 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 the 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 adding a 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 wage gap, used in the tickets regarding 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 accomodation, 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 handcrafted.

3.2.2 Bayesian Network

A Bayesian Model is a machine learning algorithm that is built with Bayes Theorem in mind. It is based on the probability theory and it enables a machine to learn from the data.

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{1}$$

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

- A feature for each node i appartenente a 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 there are also implementations that consider also continuous variables [https://www.jair.org/index.php/jair/article/download/11063/26242/]

Building a Bayesian network starting from the *Absenteeism at work Data Set* is relatively easy, we calculate the likelihood distribution $p(x\_i|x\_j)\forall x_i, x_j \in D$ for each other feature x_j. As a prior we used a dirichlet distribution. [https://mbernste.github.io/files/notes/Psued
https://cs.nyu.edu/ roweis/csc412-2004/notes/lec13x.pdf 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

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

where ϵ is the privacy budget for differential privacy, which controls the anonymization level.

$$Pr[M(X) \in Z] \leq e^{\epsilon} \cdot Pr[M(X') \in Z] \tag{3}$$

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 fake 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, 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's an example of a template:

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}

Dear Sir/Madame, my name is ${name} and I work at ${company}. I am requesting <*generate*>. I hope <*generate*>.

The variables are replaced with the features of the employee, whereas the <*generate*> are replaced with text generated by a generative model. The part generated by a pre-trained model are used in a recursive way. 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 prompt the entire ticket, including the text generated previously. The model is forced to generate some text, if no text in an iteration the generation is repeated until the model gives an output not empty.

3.4 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 is 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 gpt generation
- *max_length*: the max length of the sequence to be generated.
- *top_k*: the k most likely next words are filtered and the probability mass is redistributed among only these k words.
- *top_p*: the next words are sampled from the smallest possible set of words whose cumulative probability exceeds the probability p.
- *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

$$p_i = \frac{exp(x_i/T)}{\sum_j exp(x_j/T)} \tag{4}$$

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

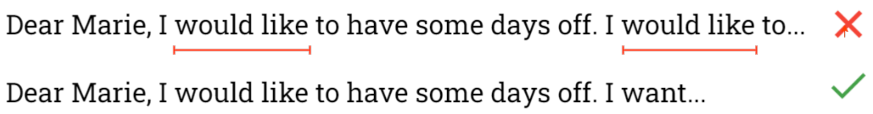
$$p_i = \frac{exp(x_i/T \cdot I(i \in G))}{\sum_j exp(x_j/T \cdot I(j \in G))} \tag{5}$$

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

  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.

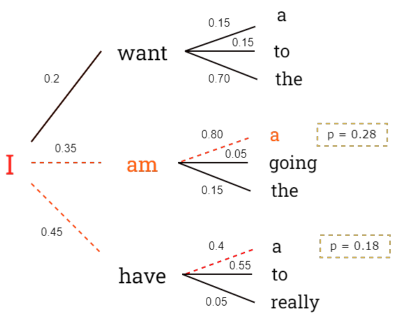
- *no_repeat_ngram_size*: If set to int > 0, all ngrams of that size can only occur once



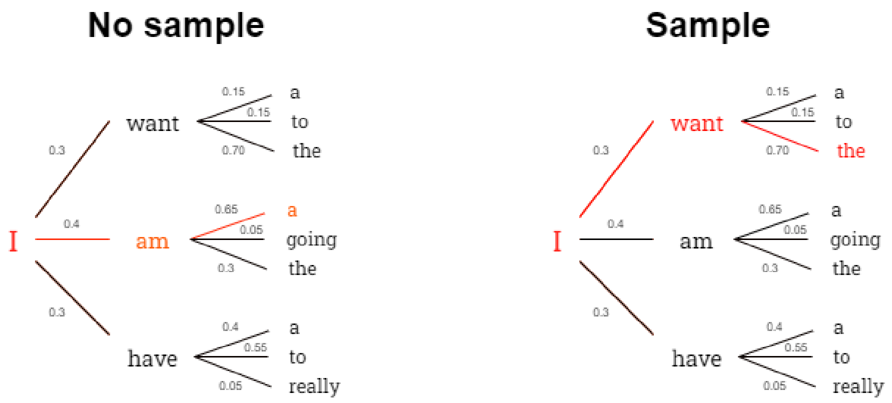
Dear Marie, I would like to have some days off. I would like to... ✗

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



- *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 2

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 2: Parameters of GPT-J model

The GPT architecture is based on the original Transformers paper, which introduced two types of transformers blocks: the encoder block and the decoder block. GPT is assembled by a stack of decoder blocks, which are composed by:

- Normalization Layers: a normalization layer [1] 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 [2]

  x: data sample
  d: dimension of data sample

y: output of LayerNorm
ϵ: small number added for stability

$$u = \frac{1}{d}\sum_{i=1}^{d} x_i \tag{6}$$

$$\sigma^2 = \frac{1}{d}\sum_{i=1}^{d} (x_-ui)^2 \tag{7}$$

$$\hat{x_i} = \frac{x_i - u}{\sqrt{\sigma^2 + \epsilon}} \tag{8}$$

$$y = \gamma\hat{x_i} + \beta \tag{9}$$

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

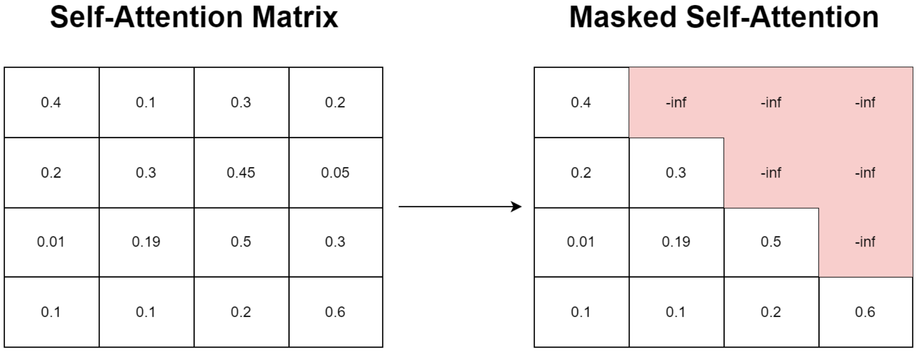
  The Q, K and V matrixes are initialized as:
  - $Q = W_qX + b_q$
  - $K = W_kX + b_k$
  - $V = W_vX + b_v$

  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

  $$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{10}$$

  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 know any information regarding the tokens at the next positions. This is fundemantel when training generative models such as GPT, whose scope is to predict the successive tokens.

  $$Attention(Q, K, V) = softmax\left(\frac{QK^T + Mask}{\sqrt{d}}\right)V \tag{11}$$



Self-Attention Matrix

0.4	0.1	0.3	0.2
0.2	0.3	0.45	0.05
0.01	0.19	0.5	0.3
0.1	0.1	0.2	0.6

Masked Self-Attention

0.4	-inf	-inf	-inf
0.2	0.3	-inf	-inf
0.01	0.19	0.5	-inf
0.1	0.1	0.2	0.6

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

Compared to GPT-3, GPT-J has two minor architectural differences[3] (shown in Figure 1):

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

3.4.1 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

$$\begin{cases} p_{1,2t} = sin(k/10000^{2t/d}) \\ p_{1,2t+1} = cos(k/10000^{2t/d}) \end{cases} \tag{12}$$

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 multiplyng the context representation with the sinusoidal functions instead if directly adding them.
If we define

$$\begin{aligned} q_m &= f_q(x_m, m) \\ k_n &= f_k(x_n, n) \end{aligned} \tag{13}$$

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 $m - n$

$$\langle f_q(x_m, m)\ ,\ f_k(x_n, n)\rangle = g(x_m, x_n, m - n) \tag{14}$$

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

$$f_{(\{q,k\},\{m,n\})} = (W_{\{q,k\}} x_{\{m,n\}}) e^{i\{m,n\}\theta} \tag{15}$$

and therefore we obtain

$$g(x_m, x_n, m - n) = Re[(W_q x_m)(W_k x_n)^T e^{i(m-n)\theta}] \tag{16}$$

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

$$Attention(Q, K, V) = softmax\left(\frac{g(x_m, x_n, m - n)}{\sqrt{d}}\right) V \tag{17}$$

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[4], using RoTE leads to a faster convergence of training and validation losses and a lower overall validation loss.

3.5 Survey

3.6 Results

3.7 Additional Experiments

4 Use Cases

4.1 Classification

4.2 NER

4.3 Anonymization

4.4 Summarization

5 Conclusion

References

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[2] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin, “Understanding and improving layer normalization,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[3] B. Wang and A. Komatsuzaki, “GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model.” `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

[4] S. Biderman, S. Black, C. Foster, L. Gao, E. Hallahan, H. He, B. Wang, and P. Wang, “Rotary embeddings: A relative revolution.” `blog.eleuther.ai/`, 2021. [Online; accessed].

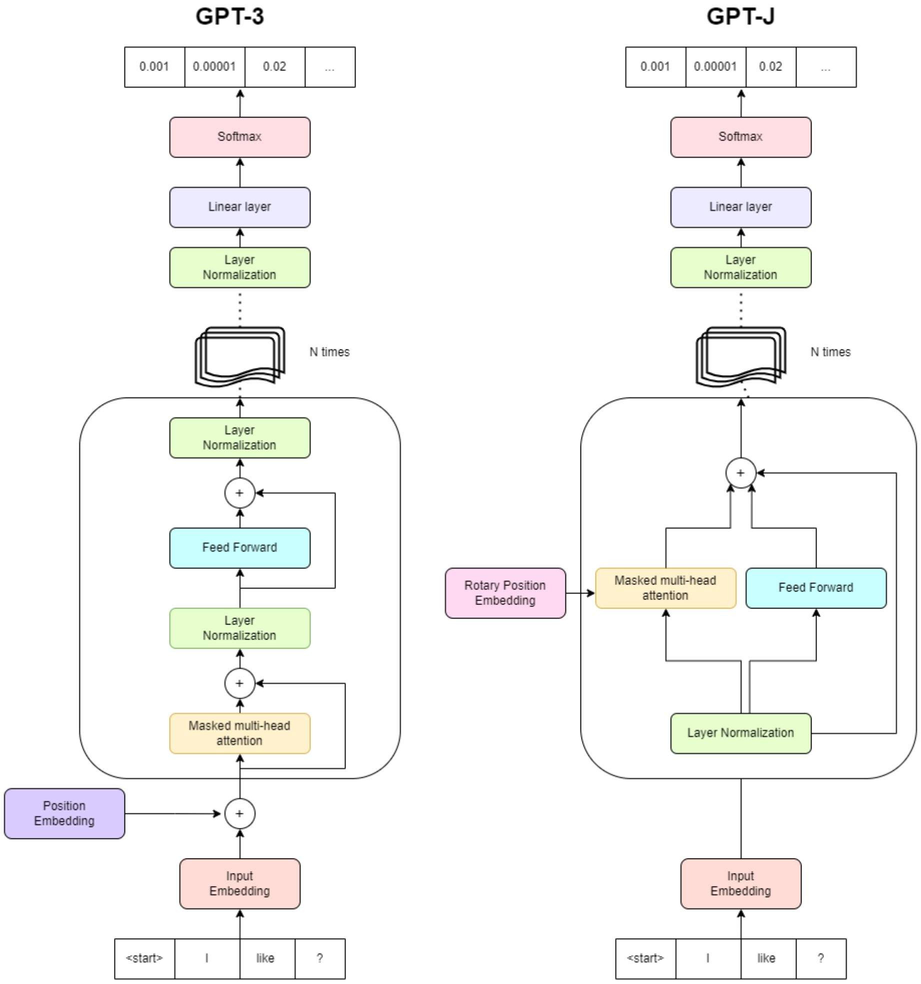


Figure 1: GPT-3 and GPT-J architectures compared