# Introduction

Difference of written summary and embedding summary.

Learning summaries of text is a fundamental problem in natural language processing, and of special interest in the context of Large Language Models (LLMs). While current approaches aim at summarizing text with a written summary that is shorter than the original text, we propose a novel approach, which reverse engineers these summaries. Rather than prompting a LLM to write a summary of a given text, we perform an optimization to find the input embedding to this LLM, which maximizes the likelihood for the LLM to generate the given text.

We aim to link these summary embeddings to consumer preferences, and plan to use these data to inform the generation of new claims for specific contexts.

One way how such a generation could work, is by averaging the embeddings of a set of claims, and then using this average embedding to generate a new claim.

We could use Combinatorial Bandits to learn about how to create linear combinations of existing embeddings, such that the generated new claim is more appealing to the consumer.

A more sophisticated approach would lie in the use of Multi-Armed Bandits, where we set the arms to be certain elements of the embedding vector. By creating a stack of MAB and LLM, we could give the MAB the ability to generate new claims and to learn about the preferences of the consumer for these new claims. In this, our summary embeddings play a key role, as they allow us to generate new claims in a deterministic fashion, which in-turn enables us to learn about the preferences of the consumer for these new claims.

Another empirical applications could lie in learning about consumer preferences for headlines of news articles or posts, and to use these data to inform the generation of new headlines.

There are ample use cases and possible empirical investigations in a marketing context for these summary embeddings. In the following, we want to highlight three different applications, that might deliver novel insights due to the use of these summary embeddings.

First, we want to research how marketers design choices on the creation of product claims relates to consumer preferences. For example, a marketer might decide between making a more factual or a more emotional claim. While there are theoretical motivations for these types of decisions, it would allow us to quantitatively assess these styles of claims and to learn how consumer preferences relate to them. An extension of this research could lie in the relation of human marketers and the help of AI, namely LLMs, in the creation of these claims. Investigating how AI and human generated claims that are supposed to fulfill a certain characteristic, such as being more emotional, are actually perceived as such by consumers, could deliver novel insights.

Second, in a more field-based setting, we want to investigate how we can use these summary embeddings to adapt online descriptions to the preferences of the consumer. While there are techniques based on Reinforcement Learning, which is used to adapt the website to the preferences of the consumer, already in use, these types of algorithms cannot make adaptations “out-of-sample”, i.e. they cannot generate new content that is not already predefined. We believe that the use of summary embeddings could allow us to generate adaptive text-based content, such as product descriptions or article headlines, that are not part of a predefined set.

Third,

In preliminary tests, we have found that these summary embeddings appear to be able to differentiate different styles of claims in the same domain.

# Methods

* Novel, “reverse engineering” approach to finding text summaries (find a better word… in an information sense, not in a writing short summary sense). Given a target sequence and a given LLM, we want to find the input embedding to this LLM, which maximizes the likelihood for the LLM to generate the target sequence.
* Training can be done with simply forward passes through the LLM model

## Problem Formulation

For a given Large Language Model, , we want to find the input embedding, , that maximizes the likelihood of generating a given target sequence, , where denotes the length of the target sequence and represents token .

To express that we generate tokens with the LLM, based on length input-embedding , we use the notation . To express the number of tokens, that we generate, we use the subscript . This function call returns an output vector of length , where denotes the vocabulary size. We denote the output vector as . Each element of represents the probability of generating token as the next token. We predict the next token, by selecting the largest element of . Explicitly, $t\_{1} = \argmax \mathcal{M}\_1(\boldsymbol{e})$.

When , we use the previous tokens to predict the next token. When , we append the predicted tokens autoregressively to the input embedding, such that we can predict more than one token. While there are many ways to design this procedure and select tokens along the way, we perform a standard greedy procedure, where we select the token with the highest probability at each step.

We denote the probability that generates the target sequence, given the input embedding, as .

We estimate these embeddings, by maximizing the conditional likelihood that this embedding generates the target sequence for a given LLM. We use a gradient-based optimization algorithm to maximize this likelihood.

These embeddings have the same dimension as the input embedding of , length .

$$
\boldsymbol{s^{\*}} = \argmax\_{\boldsymbol{s}} \prod\_{i=1}^{L} p(t\_1, \ldots, t\_L | \boldsymbol{s})
$$

$$
\boldsymbol{s^{\*}} = \argmax\_{\boldsymbol{s}} \sum\_{i=1}^{L} \log p(t\_1, \ldots, t\_L | \boldsymbol{s})
$$

$$
\boldsymbol{s^{\*}} = \argmax\_{\boldsymbol{s}} \log p(t\_1, | \boldsymbol{s}) + \log p(t\_2, | \boldsymbol{s}, t\_1) + \ldots + \log p(t\_L, | \boldsymbol{s}, t\_1, \ldots, t\_{L-1})
$$

$$
\boldsymbol{s^{\*}} = \argmax\_{\boldsymbol{s}} \log \mathcal{M}\_1(\boldsymbol{s}) + \log \mathcal{M}\_1(\left [\boldsymbol{s}, \boldsymbol{e}\_1 \right]) + \ldots + \log \mathcal{M}\_1(\left [\boldsymbol{s}, \boldsymbol{e}\_1, \ldots, \boldsymbol{e}\_{L-1} \right])
$$

$$
\boldsymbol{s^{\*}} = \argmax\_{\boldsymbol{s}} \sum\_{i=1}^{L} \log \mathcal{M}\_L (\boldsymbol{s})
$$

We can make this expression more explicit, by using the LLM in our notation. For example

In the training process, we execute the following algorithm. First, we initialize the input embedding, . Second, we generate a sequence of length , with the LLM and the current input embedding.