Experiment Maker: a Tool to create Experiments with GPT-3 easily*

Patrizio Bellan[1,2,*], Mauro Dragoni[2] and Chiara Ghidini[2]

[1]Fondazione Bruno Kessler, Via Sommarive 18, 38123, Trento, Italy
[2]Free University of Bozen-Bolzano, Bolzano, Italy

Abstract

Novel large pre-trained language models, such as GPT-3, can be considered and adopted as artificial agents since they are now able to solve general problems and mimic human experts. The introduction of the in-context learning technique allows interaction with the model directly by instructing it to solve a task. Task instructions, the actual input data, and optionally some examples of solutions are packed together in a single prompt. The model interprets the prompt and generates a solution for the given problem without any need of fine-tuning the model.

However, designing efficient prompts is more an art than a science, nowadays. When starting from scratch, to achieve good performance different prompt contents and model engine's configurations (for the same prompt) must be tested. These can be considered time-intensive operations.

In this paper, we present *Experiment Maker* a software developed to save time and minimize the effort in designing and testing different prompts and different configurations, In addition, the tool supports users in combining multiple prompts into an experimental pipeline. *Experiment Maker* can be downloaded from the project page at github.com/patriziobellan86/ExperimentMaker.

Keywords

GPT-3, in-context learning, python, experiments

1. Introduction

Nowadays, many real-world scenarios such as Process Extraction from Text [1] may not be supported by transformer-like models, such as BERT [2] or RoBERTa [3] in a classical training manner. The data available to train these models toward a downstream task is too small in size. While the fine-tuning strategy for task-specific applications is standard practice in NLP, the advent of GPT-3 [4] has greatly changed this paradigm. This model opens the possibility of injecting task-specific instructions into a model without altering any model's weight or bias. This technique is called **in-context learning**. Task instructions, the actual input data, and optionally some examples of the task to solve are fed all together in a *prompt*. Then the prompt is fed into the model and the model generates the solution. This approach has been widely adopted to address topics ranging from medical dialogue summarization [5] to hate speech detection [6].

EKAW'22: Companion Proceedings of the 23rd International Conference on Knowledge Engineering and Knowledge Management, September 26–29, 2022, Bozen-Bolzano, IT

*Corresponding author.

pbellan@fbk.eu (P. Bellan); dragoni@fbk.eu (M. Dragoni); ghidini@fbk.eu (C. Ghidini)
0000-0002-2971-1872 (P. Bellan); 0000-0003-0380-6571 (M. Dragoni); 0000-0003-1563-4965 (C. Ghidini)

Applying in-context learning technique with Large Pre-Trained Language Models (PLMs), such as GPT-3, is difficult since many aspects may impact the performance of a given task. Indeed, task performance not only is heavily dependent on the textual content and formatting style of the prompt but it is also conditioned by the model engine configuration. The most important model engine parameter to set is *the sampling strategy* since its value has a direct impact on task accuracy. Let the model be too "creative" may solve some tasks but, in general, hamper significantly the performance. To gain good performance for a task, different prompt contents and model engine configurations must be tested. This calibration step is a time-intensive operation because it must be done for each prompt. In addition, some problems may be too complex to be solved with a single prompt. Complex problems can be divided into small sequential sub-problems, each of which could be solved by a specific prompt. So, prompts can be combined all together in a pipeline to solve complex tasks. Creating a pipeline is not an easy task that could become a very intensive operation.

In this paper we present *Experiment Maker*, a software developed to reduce the effort and the time spent when design prompts and combining multiple prompts in a pipeline. Our system is composed of two complementary components: *Prompt Designer* and *Pipeline Maker*. *Prompt Designer* support users to design and calibrate prompts. *Pipeline Maker* helps users in creating a pipeline by combining multiple prompts. It also allows using the results of a previous step as actual input data of another prompt. This component allows users to execute custom python modules (typically used as data filters) at the end of both each step and the entire pipeline. Custom modules help in automatizing some common tasks along a pipeline execution, such as extracting items from results or cleaning them. Practically, they allow tailoring pipelines toward specific needs, tasks, domains, and output formats.

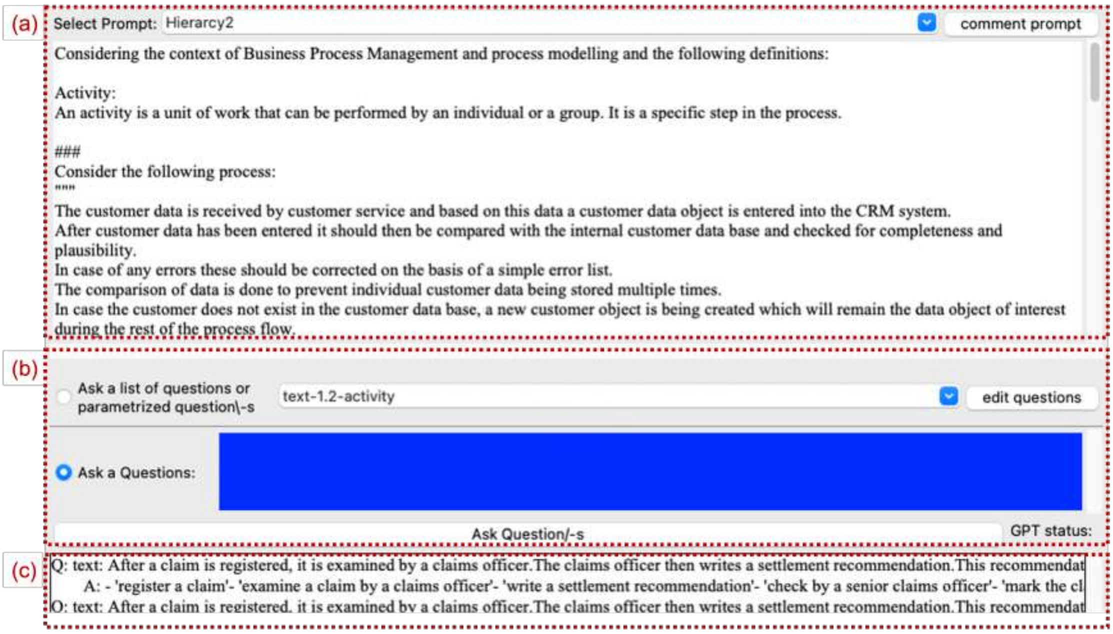
2. System Implementation

The system relies on two main components: *Prompt Designer* and *Pipeline Maker*.

***Prompt Designer*.** This component supports the design and testing of prompts. When designing prompt two important aspects have to be considered: (i) prompt content and (ii) model engine configuration. A straightforward aspect to consider when designing a prompt from scratch is the selection of information to provide to the model. For example, it is crucial to test the same prompt with different examples of the task to solve, or different task instructions since they have a great impact on the results. During the preliminary tests, different customizations must be tested to observe the inference capability of the system and so tuning the queries. This type of test is necessary to observe how much the content or/and the formatting style of the prompt may affect the results. Different configurations lead to different results.

To test automatically different prompt contents, such as different task instructions or different examples of the task, *Prompt Designer* allows the creation of custom place-holders. Placeholders will be filled with a specific list of values provided directly in the editor or read from a file. This aspect allows testing the same prompt content, filled with different task instructions, for example.

One of the main advantages of the GPT-3 model is the possibility to adjust the model engine



Select Prompt: Hierarcy2

```
Considering the context of Business Process Management and process modelling and the following definitions:

Activity:
An activity is a unit of work that can be performed by an individual or a group. It is a specific step in the process.

###
Consider the following process:
"""
The customer data is received by customer service and based on this data a customer data object is entered into the CRM system.
After customer data has been entered it should then be compared with the internal customer data base and checked for completeness and plausibility.
In case of any errors these should be corrected on the basis of a simple error list.
The comparison of data is done to prevent individual customer data being stored multiple times.
In case the customer does not exist in the customer data base, a new customer object is being created which will remain the data object of interest during the rest of the process flow.
```

○ Ask a list of questions or parametrized question\-s: text-1.2-activity

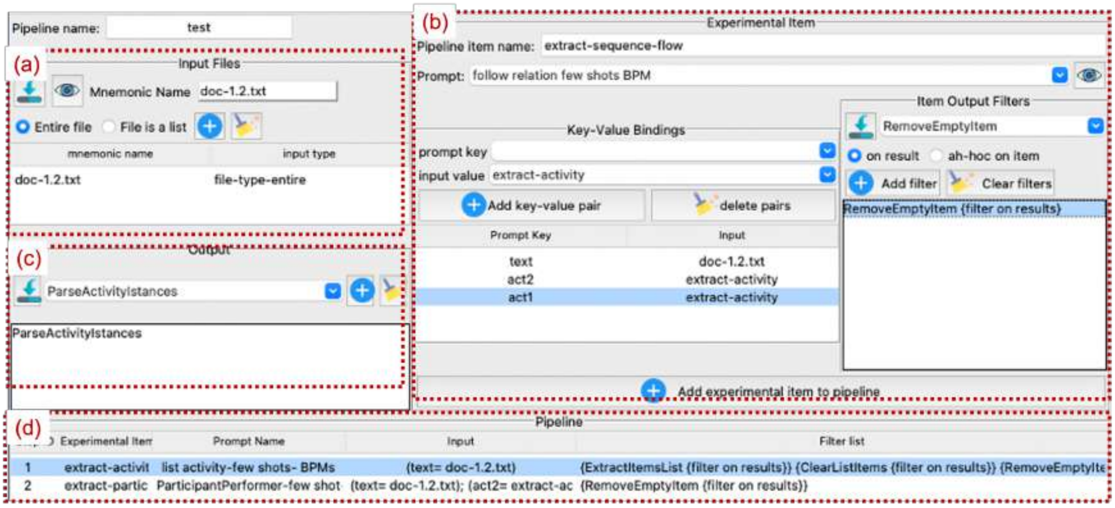
● Ask a Questions:

```
Q: text: After a claim is registered, it is examined by a claims officer.The claims officer then writes a settlement recommendation.This recommendat
      A: - 'register a claim'- 'examine a claim by a claims officer'- 'write a settlement recommendation'- 'check by a senior claims officer'- 'mark the cl
Q: text: After a claim is registered, it is examined by a claims officer.The claims officer then writes a settlement recommendation.This recommendat
```

Figure 1: The figure shows a screen-shot of *Prompt Designer* component.

configuration. The crucial parameter to set is the token sampling strategy. This parameter controls the level of "creativity" of the model in solving tasks. This is where *temperature* and *nucleus* sampling strategies come into play. These two parameters control how confident the model should be when sampling tokens from the tokens' distribution. Increasing one of these two parameters will make the model more likely to take "risks" and it will consider tokens with lower probabilities. Finally, *presence penalty* and *frequency penalty* parameters allow for conditioning of the responses generated by allowing the model to sample new words and avoid repetitions, for example. To support the user in understanding the impact of different prompt contents/model engine configurations, *Prompt Designer* offers a dedicated comparison window that allows selecting the answers to show, i.e., comparing different prompt contents or different model engine configurations. A screenshot of the user interface is shown in Fig.1. The GUI is composed of three main parts. The first part (a) is a *prompt editor*. Here it is possible to edit the prompt content and add placeholders. In the second part of the GUI (b) the user can choose between (i) asking a single question to the language model or (ii) binding the placeholders defined in the prompt with variables. Finally, in the third part of the GUI (c), the interface reports a list of the request posed to the language model.

Pipeline Maker. This component helps users to combine prompts to achieve a common goal or solve a complex problem. The extraction of a knowledge graph out of a text could be an example of a complex problem. Indeed, the extraction could involve the execution of a sequence of sub-tasks. Each sub-task can be solved by a prompt along a pipeline (a sequence of consecutive steps). An example of such a pipeline could be: (i) extracts entities described; (ii) infers entities



Pipeline name: test

Mnemonic Name: doc-1.2.txt

● Entire file ○ File is a list

mnemonic name	input type
doc-1.2.txt	file-type-entire

ParseActivityIstances

ParseActivityIstances

Pipeline item name: extract-sequence-flow

Prompt: follow relation few shots BPM

prompt key:

input value: extract-activity

Prompt Key	Input
text	doc-1.2.txt
act2	extract-activity
act1	extract-activity

RemoveEmptyItem

● on result ○ ah-hoc on item

RemoveEmptyItem {filter on results}

D	Experimental Iterr	Prompt Name	Input	Filter list
1	extract-activit	list activity-few shots- BPMs	(text= doc-1.2.txt)	{ExtractItemsList {filter on results}} {ClearListItems {filter on results}} {RemoveEmptyIte
2	extract-partic	ParticipantPerformer-few shot	(text= doc-1.2.txt); (act2= extract-ac	{RemoveEmptyItem {filter on results}}

Figure 2: The figure shows a screen-shot of *Pipeline Maker* component.

not described; (iii) infers their relations. The advantage of *Pipeline Maker* is the possibility to bind prompt-template place-holders with the output of a previous step. A sequence of prompts is combined into a single pipeline. Thus, *Pipeline Maker* easier the task of creating and testing pipelines.

At the end of each step, it is possible to execute a custom module to process the step results. Custom modules (called *step filter*) commonly operate as a filter on step results or are adopted to automatize some common tasks (such as item extraction and text cleaning). Finally, at the end of the entire pipeline, it is possible to execute a custom module, called *output filter*, to perform the final filtering of the results (if needed) or to generate custom representations of the overall pipeline, such as generating a knowledge graph in a specific format.

A screenshot of the user interface is shown in Fig.2. The GUI is composed of four main parts. The first part of the GUI (a) allows the user to load textual content of files and use it as a variable to be bind to prompt placeholders. In the second part (b), the user binds prompt placeholders with variables and selects the step filters to eventually apply at the end of the step. In the third part of the GUI (c), the user can load and select which filters apply at the end of the pipeline. Finally, the last part of the GUI (d) presents the workflow of the pipeline.

3. *Experiment Maker* in Action: What we Will Show During The Demo

The demo presentation will be split into two parts: (i) the presentation of the *Prompt Designer* component to design prompt, and (ii) the presentation of the *Pipeline Maker* component to combine prompts to create an experimental pipeline.

We will consider the problem of extracting a simple knowledge graph out of texts of the PET dataset[7] as a use case. We will show how to create prompts from scratch, create custom filters to manipulate the model's answers, and create an experimental pipeline entirely based on GPT-3 and in-context learning techniques.

Usage of *Prompt Designer*. In the first part of the demonstration, we will show participants *Prompt Designer* in action. We will show how to design a prompt from scratch to solve a particular task, i.e., the extraction of activities and the actor performing such activities described in a text. Moreover, we will guide participants in the calibration of model engine configuration. We will show how to compare different model engine configurations to understand the impact of the engine parameters on task performance.

Usage of *Pipeline Maker*. In the second part of the demonstration, we will show *Pipeline Maker* in action.

Firstly, a more technical demonstration is related to the creation of custom modules to automatize some steps of the pipeline and tailor the pipeline to specific needs. We will show how developers can create a custom (python) module to extract items out of GPT-3 answers and instantiate entities and relations into an ontology.

Then, we will demonstrate the creation of an experimental pipeline combining multiple prompts and custom modules. We will show how to connect multiple prompts in a chain. The results of a previous step could fill the actual input data placeholder in another step along the pipeline. We will also show how to execute the custom modules created, at the end of each step and at the end of the entire pipeline. The pipeline presented would solve the task by simulating a multi-turn dialog with a domain expert.

4. Conclusion

This paper aims at showing the advantage of using *Experiment Maker* to create experiments based on in-context learning and GPT-3 language model. Specifically, *Experiment Maker* minimizes the problem of designing prompts, calibrating the model engine's configuration toward better performance, and creating experimental pipelines to solve complex tasks by combining multiple prompts and running custom modules.

The proposed system is composed of two main components. *Prompt Designer* supports users in designing prompts and calibrates the model engine's configuration.

Pipeline Maker helps users to combine prompts and to create a pipeline. Also, this component allows the user to run a set of custom modules (written in python) at the end of every step and at the end of the pipeline.

In future developments, we plan to enable support for other PLMs supporting in-context learning.

References

[1] P. Bellan, M. Dragoni, C. Ghidini, Process extraction from text: state of the art and challenges for the future, CoRR abs/2110.03754 (2021). URL: https://arxiv.org/abs/2110.03754. arXiv:2110.03754.

[2] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Proc. of NAACL-HLT 2019, Volume 1, ACL, 2019, pp. 4171–4186.

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, RoBERTa: A Robustly Optimized BERT Pretraining Approach, ArXiv abs/1907.11692 (2019).

[4] T. B. Brown, et al., Language models are few-shot learners, in: Annual Conf. on Neural Information Processing Systems 2020, NeurIPS 2020, 2020.

[5] B. Chintagunta, N. Katariya, X. Amatriain, A. Kannan, Medically aware gpt-3 as a data generator for medical dialogue summarization, in: Proc. of the 6th Machine Learning for Healthcare Conf., volume 149 of *Proc. of Machine Learning Research*, PMLR, 2021, pp. 354–372.

[6] S. Gupta, Hate speech detection using openai and gpt-3, International Journal of Emerging Technology and Advanced Engineering (2022).

[7] P. Bellan, H. van der Aa, M. Dragoni, C. Ghidini, S. P. Ponzetto, Pet: An annotated dataset for process extraction from natural language text tasks (2022).