TÉCNICO LISBOA

Learning to Disambiguate Queries with Respect to Business Processes

Lucas Fortunato Das Neves

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Alessandro Gianola
Prof. Chrysoula Zerva

Examination Committee

Chairperson: Prof. Andreas Miroslaus Wichert
Supervisor: Prof. Alessandro Gianola
Member of the Committee: Prof. Marco Montali

November 2025

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

This work was supported by the 'OptiGov' project, with ref. n. 2024.07385.IACDC (DOI: 10.54499/2024.07385.IACDC), fully funded by the 'Plano de Recuperação e Resiliência' (PRR) under the investment 'RE-C05-i08 - Ciência Mais Digital' (measure 'RE-C05-i08.m04'), framed within the financing agreement signed between the 'Estrutura de Missão Recuperar Portugal' (EMRP) and Fundação para a Ciência e a Tecnologia, I.P. (FCT) as an intermediary beneficiary.

I would like to thank my mother for her friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible.

I would also like to acknowledge my dissertation supervisors Prof. Alessandro Gianola and Prof. Chrysoula Zerva for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends, mentors and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. To each and every one of you – Thank you.

Abstract

Process mining plays a critical role in extracting insights from organizational processes by analyzing event log data. However, it requires the engagement of domain experts and process analysts. While Large Language Models have enabled conversational agents for process mining tasks —reducing dependence on analysts — their adoption introduces new challenges. Users without an understanding of process mining often formulate ambiguous or ill-defined queries due to limited specific knowledge, hindering accurate analysis. To address this, we present an approach that integrates Retrieval-Augmented Generation Large Language Models and Chain-of-Thought reasoning. By retrieving a given event log and by reducing ambiguity with reasoning steps, human interaction with conversational agents becomes more intuitive, further bridging the existing gap. This work details the design, methodology, and implementation of the system, offering a novel framework for tackling ambiguity in process mining, thus improving user query resolution while fostering more intuitive user interaction. We also introduce datasets Query-PM-LLM and AmbQuery-PM-LLM, which can be used as benchmarks for future conversational agents capable of solving process mining tasks.

Keywords

Large Language Models; Retrieval Augmented Generation; Chain of Thought; Ambiguity; Business Process; Process Mining;

Resumo

A mineração de processos desempenha um papel fundamental na extração de insights dos processos organizacionais, através da análise de dados de registos de eventos. No entanto, requer o envolvimento de especialistas de um dado domínio e analistas de processos. Embora os Modelos de Linguagem de Grande Escala tenham possibilitado a criação de agentes conversacionais para tarefas de mineração de processos — reduzindo a dependência em analistas —, a sua adoção traz novos desafios. Os utilizadores sem compreensão da mineração de processos muitas vezes formulam questões ambíguas ou mal definidas devido ao conhecimento específico limitado, dificultando a análise precisa. Para resolver isso, apresentamos uma abordagem que integra Modelos de Linguagem de Grande Escala com Geração Aumentada por Recuperação e raciocínio em Cadeia de Pensamento. Ao recuperar registo de eventos e reduzir a ambiguidade com as etapas de raciocínio, a interação humana com agentes conversacionais torna-se mais intuitiva, preenchendo ainda mais a lacuna existente. Este trabalho detalha o design, a metodologia e a implementação do sistema, oferecendo uma estrutura inovadora para lidar com a ambiguidade na mineração de processos, melhorando assim a resolução de consultas dos utilizadores e promovendo uma interação mais intuitiva. Também apresentamos os conjuntos de dados Query-PM-LLM e AmbQuery-PM-LLM, que podem ser usados como referências para futuros agentes conversacionais capazes de resolver tarefas de mineração de processos.

Palavras Chave

Modelos de Linguagem de Grande Escala; Geração Aumentada por Recuperação; Cadeia de Pensamento; Ambiguidade; Processo de Negócios; Mineração de Processos;

Contents

List of Figures

List of Tables

Listings

Acronyms

BPM	Business Process Management
AI	Artificial Intelligence
LLM	Large Language Model
RAG-LLM	Retrieval-Augmented Generation Large Language Models
CoT	Chain-of-Thought
NLP	Natural Language Processing
BPR	Business Process Reengineering
BPMN	Business Process Model and Notation
DFG	Directly-Follows Graph
LTLf	Linear Temporal Logic on Finite Traces
SLM	Statistical Language Model
NN	Neural Network
FFNN	Feed Forward Neural Network
MLP	Multi-Layer Perceptron
RNN	Recurrent Neural Network
GRU	Gated Recurrent Units
LSTM	Long Short-Term Memory
PLM	Pre-trained Language Model
ERP	Enterprise Resource Planning
XES	eXtensible Event Stream
LCS	Longest Common Subsequence

1

Introduction

Contents

Business processes are structured sets of activities or tasks performed by individuals, teams, or systems to achieve a specific organizational goal. They are the backbone of how organizations deliver value, whether it be producing goods, offering services, or achieving internal objectives. To improve their processes, organizations often rely on a field called Business Process Management (BPM) [6]. BPM provides the methodology and tools to analyze, design, implement, monitor, and optimize business processes for greater efficiency and alignment with business objectives. To effectively map and communicate processes, organizations leverage process modelling languages with some of them prioritizing control (Figure 2.1) and others prioritizing flexibility. Despite the advantages associated with modeling business processes, there exists a disconnect between how processes are documented and how they are actually executed since the modeling fails to capture inherent deviations of real-world workflows. Consequently, organizations also rely on data to uncover the actual process execution, this way leveraging process mining [7]. Process mining analyzes event logs to infer process behavior, identify deviations from models, and derive

actionable insights for improvement.

While business processes are useful in a multitude of sectors like healthcare [8], they require knowledge about modeling concepts that domain experts tend to lack, making it more challenging for those experts to extract information from overly complex models inherent in such sectors [9]. On the other hand, process analysts, who know the modeling component, know very little about the domain being modeled. As a result, extracting valuable insights from process analysis requires the engagement of multiple stakeholders, which often leads to delays and miscommunication [10].

Large Language Models (LLMs) have revolutionized Artificial Intelligence (AI), enabling advanced understanding and generation of human language [11]. Based on the Transformer architecture [12] and trained on vast datasets, LLMs are versatile tools influencing diverse fields like healthcare, education, business, and the arts. Given this rise of LLMs, in order to bridge the gap in terms of skill and allow domain experts to make better use of business processes, conversational agents that can solve specific process mining tasks have already been developed [4, 13]. While those conversational agents are helpful for business experts, they introduce a new set of challenges. LLMs, despite their advanced capabilities, are prone to the inherent ambiguity of natural language. This ambiguity can result in misinterpretations, hallucinations, or biased responses, further complicating the analysis [14]. For example, if a user prompts a model with "Could you enumerate the activities that occur in the Sepsis Cases - Event Log?", the model knows it has to provide a list of activities; however, if the user prompts the same model with "Tell me what happens in the sepsis data.", the model might be uncertain about the type of the expected answer (i.e., the user might desire the list of activities that happen in the Sepsis Cases - Event Log or a list of conformant traces that exemplify typical behavior).

Consequently, while these tools bridge skill gaps and enhance accessibility for domain experts, they also highlight the need for more robust strategies to manage and mitigate the effects of ambiguity in Natural Language Processing (NLP) when solving process mining tasks.

1.1 Work Objectives

Aimed at disambiguating queries provided by non-expert users who may lack the technical knowledge, we propose a system that can increase the robustness of conversational agents that assist in process mining tasks. This robustness will come from the deployment of AI techniques that can help retrieve and analyze event logs from external databases - using a naive approach to Retrieval-Augmented Generation Large Language Models (RAG-LLM) - and disambiguate user queries by relying on reasoning – using Chain-of-Thought (CoT).

Segmenting it into multiple objectives:

- O1: prepare a dataset that can be used to measure the model's behaviour on ambiguous questions.

- O2: implement a naive RAG-LLM framework that retrieves and processes the relevant event log to support answer generation for a given user query.
- O3: experiment with CoT by providing the model with the reasoning steps that can help disambiguate a user query and provide the final answer, verifying if reasoning can help deal with ambiguity.

1.2 Expected Contributions

Given the multitude of ways that LLMs are being applied in this segment of study, some of the expected contributions consist of:

- EC1: a conversational agent for process mining that is capable of dealing with ambiguity.
- EC2: a new approach to deal with ambiguity in natural language by relying on CoT.
- EC3: new datasets that can serve as a benchmark for future work involving tailored NLP tasks in the context of process mining.

1.3 Document Structure

This document begins with Chapter 2, which provides background information and introduces concepts necessary for understanding the subsequent chapters. Chapter 3 presents related work that inspires the proposed system and highlights the differences from our own approach, which is described in detail in Chapter 4. In Chapter 5, the metrics used to evaluate the proposed system are introduced, followed by Chapter 6, where the performance of the system with respect to these metrics is discussed. Finally, Chapter 7 concludes the document and outlines directions for future work.

2

Background

Contents

In this Chapter, we present the concepts that are fundamental to understand the problems addressed and the implemented system. We begin with business processes, which provide the structure for how organizations operate. We then present process mining, which uses event data to uncover how these processes are actually executed. While process mining enables data-driven insights, it requires technical expertise. We explore LLMs, which can make the interaction more intuitive for non-experts, enabling them to query about process data. We finish by presenting the concept of ambiguity and its prevalence in the subjects previously described.

2.1 Business Process

In a business context, processes are essential to ensure tasks are completed in a manner that enhances cost efficiency and operational effectiveness while eliminating inconsistencies. They are fundamental to the seamless functioning of companies and naturally vary across industries, as each sector has unique requirements and priorities. The discussions and descriptions in this Section are based on the insights and methodologies presented in the book Fundamentals of Business Process Management [1].

2.1.1 Process Description

A process is a set of activities performed by multiple actors, a set of events that happen instantaneously and decision points. When an activity is simple and constitutes a single unit of work, it is called a task (e.g., checking if received equipment matches an order). If multiple steps need to be performed (e.g., instead of only checking if received equipment matches an order, also consider inspecting equipment specifications, functionality, and accessories), we refer to it as an activity instead of a task. Thus, a task is an atomic unit of work performed by one participant, while activity is a broader term encompassing both simple and complex units of work. An event corresponds to something that happens without duration or effort (e.g message received). Note that when an event happens, it can trigger the execution of multiple activities. A decision point, as the name suggests, corresponds to a moment when a decision, which will influence the execution of the process, has to be made and it is usually conditioned on the outcome of previous activities or triggering of an event. A process also involves various elements, including physical objects (e.g., equipment, materials, or paper documents), and informational objects (e.g., electronic documents or records).

For example, Figure 2.1 presents the Business Process Model and Notation (BPMN)(further described in Section 2.1.3) representation of the initial fragment of the equipment rental process. This process starts when a Site Engineer (actor) needs new equipment (event), and finishes when a process order is created (event) or the request to rent equipment is rejected (event). Between those events multiple activities (e.g., Submit equipment rental request, Review rental request) are performed depending on the progress after decision points (e.g., Review rental request is only performed if equipment is available).

2.1.2 BPM

The study of business processes is motivated by the need to optimize organizational operations, ensuring they are efficient, cost-effective, and adaptable to changing demands. By systematically analyzing and improving processes, organizations can eliminate redundancies, enhance quality, and drive competitive advantage. This section introduces BPM, a discipline that formalizes these efforts, and traces its evolution through a pivotal historical example to underscore the tangible benefits it provides.

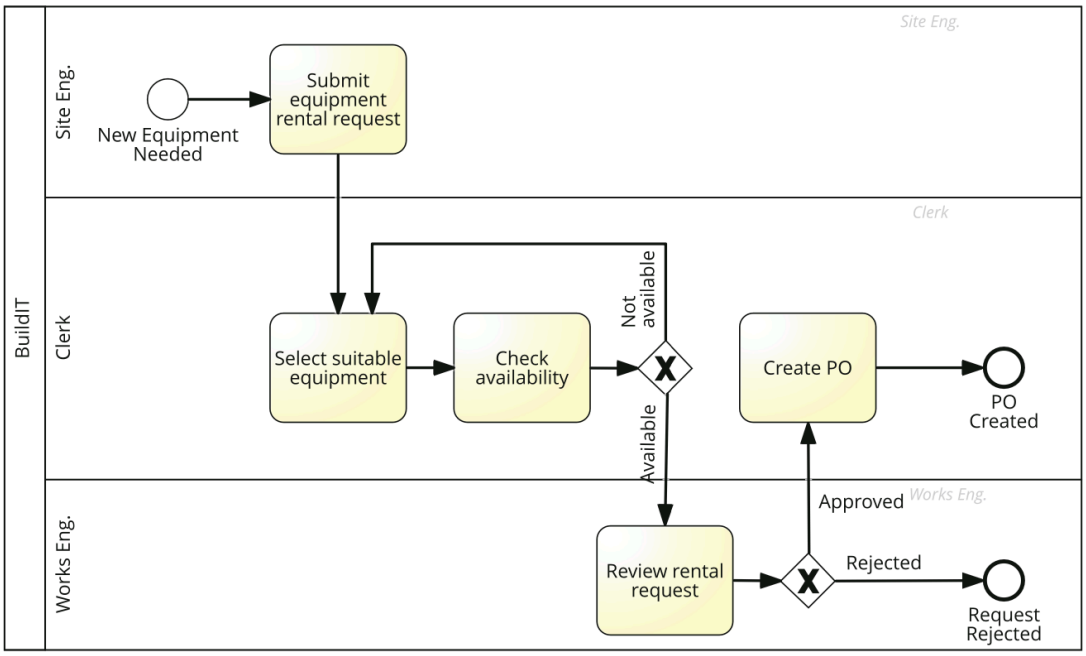

Figure 2.1: Process Model (BPMN) for the initial fragment of the equipment rental process. Source: [1]

A key event in the development of BPM occurred in 1980s, when Ford acquired a stake in Mazda and noticed that Mazda's facilities were way too understaffed to have the outcomes that they verified. This observation led to Business Process Reengineering (BPR), which focused on radically redesigning processes for improvements in cost, quality, and speed. After analysis, it was notable that Ford's purchasing process was inefficient, requiring extensive checks of documents by accounts payable. Inspired by Mazda's approach, Ford redesigned the process, adding a central database and computer terminals with access to the database, allowing immediate verification of deliveries and reducing its accounts payable staff by 76%. The case emphasized the importance of redesigning entire processes.

However, due to a myriad of reasons (e.g., concept misuse, lack of technological tools), BPR did not last many years. It was only later on, when there were technical studies available on the value it had for companies and with the technological advancement that it came back rebranded as BPM. Today, BPM provides a structured framework for continuous process improvement as outlined in Section 2.1.2.A.

2.1.2.A BPM Lifecycle

The BPM lifecycle comes into play when there is a process that needs to be improved (Process identification). Process identification corresponds to the first phase of the lifecycle as can be seen on Figure 2.2, and it is followed by:

- Process discovery - corresponds to the documentation of the as-is process model. The as-is process model corresponds to the business process before any operational improvement and is built by the

process analyst. For that purpose, the analyst resorts to interviews, documentations and workshops which help understand how things work in the organization.

- Process analysis - through qualitative and quantitative analysis, weaknesses of the as-is processes are identified. For qualitative analysis, things like Value Added Analysis (which helps identify tasks that add no value) and Issue Registers (which is a list that helps organize and prioritize weaknesses) can be considered. For the quantitative analysis, process simulation can be done by tools like Bizagi[1].

- Process redesign - changes aimed at addressing the weaknesses of the as-is process are proposed. Those changes are analyzed with the process analysis techniques and if they yield the expected improvements, they are implemented, going from the as-is process model to the to-be process model.

- Process implementation - corresponds to making the changes that are necessary for the implementation of the to-be process model. Those changes can either be organizational change management (activities that change the way people work) or automation (with the help of IT systems).

- Process monitoring - the process is analyzed and data is collected to check if the process has the expected performance. If new weaknesses arise, the cycle repeats.

2.1.3 Process Models

Business process models are essential throughout the BPM lifecycle and must be created with a clear purpose in mind, as their design varies based on the intended use. One key reason for modeling is to understand and share insights about a process in a formalized manner, especially since participants often focus on specialized tasks and may not see the full complexity.

A process model is a formal representation of a process, encompassing the set of activities, steps and rules [10]. One of the main challenges that can be found in managing processes is the balancing between flexibility and control and that leads to two approaches:

- Procedural: Procedural models are ideal for explicitly representing control-flow patterns like sequences, decisions, parallel executions, and loops. These models effectively capture a subset of the possible execution paths (traces) but often face challenges when dealing with loosely structured or highly flexible processes [15]. In essence, they prioritize control over flexibility and are focused on specifying how a process should be executed, using imperative modeling languages like:

[1] https://www.bizagi.com/en

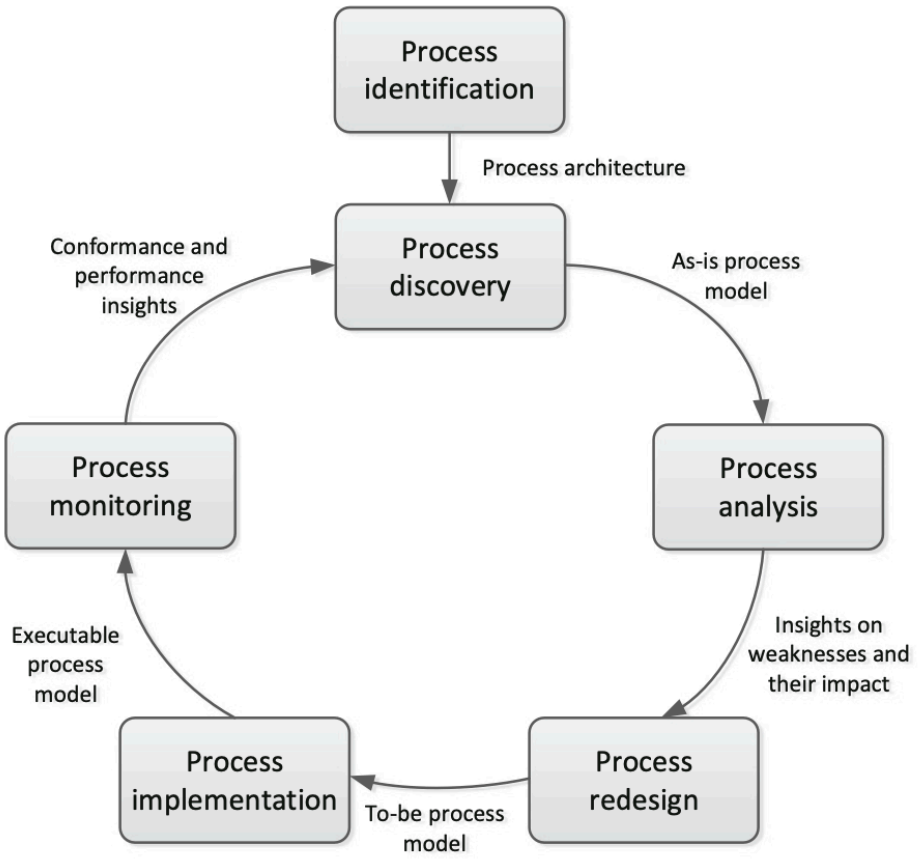

Figure 2.2: BPM Lifecycle. Source: [1]

– BPMN: allows businesses to visualize their internal processes in a clear, graphical format, enabling effective communication of these procedures in a standardized way (Figure 2.1). This graphical representation also enhances the understanding regarding collaboration and business transactions among organizations. As a result, businesses gain a deeper understanding of their operations and stakeholders[2].

– Petri Nets: A Petri Net is a formal graph model used to represent the control behavior of systems with concurrent operations. It consists of places (represented by circles) and transitions (represented by bars), connected by directed arcs[3].

– Directly-Follows Graphs (DFGs): simple representation of process behavior derived from event logs - described in Section 2.2. They focus on capturing the direct relationships between activities in a process[4].

[2] https://www.bpmn.org/
[3] https://link.springer.com/referenceworkentry/10.1007/978-0-387-09766-4_134
[4] https://www.vationventures.com/glossary/directly-follows-graph-definition-explanation-and-use-cases

- Declarative: As described in [15], declarative process modeling languages emphasize flexibility by specifying constraints on a process's temporal evolution without prescribing exact execution paths. They better define the boundaries of allowed and forbidden behaviors, enabling a more comprehensive representation of real processes. The main language of this approach is DECLARE, relying on Linear Temporal Logic on Finite Traces (LTLf) [16], which reflects the expectation that process instances conclude in a finite number of steps and allows deeper analysis and understanding of constraints and temporal relationships in a process.

2.2 Process Mining

Instead of using standard approaches to discover a process as described in Section 2.1.2.A, data can be used to discover the process automatically with the help of process mining. More formally, process mining is the practice of using event data and process models to improve operational processes, representing the intersection of Data Science and Process Science [7]. It helps identify inefficiencies, bottlenecks, and deviations, as well as diagnose performance and compliance issues. Process mining can be both backward-looking, such as uncovering the root cause of a problem, and forward-looking, like predicting processing times or suggesting improvements. It focuses on processes that require the execution of repetitive tasks, which are common in various industries like production, logistics, finance, healthcare, and government.

Process mining involves extracting event data from information systems. This data, often scattered across multiple tables, must be converted into a usable format for analysis, making the extraction process time-consuming. Once data is gathered, it is explored, filtered, and cleaned using dedicated query languages (e.g. SQL). The cleaned dataset is called an event log. This event log is a compilation of the events previously gathered, where each of them has associated with it a case identifier, an activity name, a timestamp and optional attributes [17], as presented in Table 2.1.

The case identifier refers to a case, which is an instance of the process. A list of sequential events within a case is defined as a trace. A set of traces that have identical sequences of activities corresponds to a single process variant, since process variants disregard the remaining attributes(e.g., timestamp, case identifier, resource).

With an event log, there are multiple tasks that can be performed:

- Process Discovery: based on the (event log) data, discover a process.
- Conformance Checking: given a model and an event log, find out if there is disagreement/mismatch between them.
- Performance Analysis: since it is possible to relate event data to process model, it is also possible to verify the existence of loops, frequencies of undesired activities, and activities that take too long and lead to increased waiting times.

Table 2.1: Fragment of Sepsis Cases - Event Log

Case ID	Timestamp	Activity	Resource
...	...	...	...
G	2014-09-05 14:40:47+02:00	ER Registration	A
G	2014-09-05 14:52:39+02:00	ER Triage	C
G	2014-09-05 14:53:30+02:00	ER Sepsis Triage	A
G	2014-09-05 15:19:00+02:00	IV Liquid	A
G	2014-09-05 15:19:14+02:00	IV Antibiotics	A
G	2014-09-05 15:21:00+02:00	LacticAcid	B
G	2014-09-05 15:21:00+02:00	CRP	B
G	2014-09-05 15:21:00+02:00	Leucocytes	B
G	2014-09-05 19:20:13+02:00	Admission NC	H
G	2014-09-09 08:00:00+02:00	Leucocytes	B
G	2014-09-09 08:00:00+02:00	CRP	B
G	2014-09-12 10:00:00+02:00	Release A	E
H	2014-03-11 09:50:02+01:00	ER Registration	A
H	2014-03-11 09:51:06+01:00	ER Triage	C
H	2014-03-11 09:51:26+01:00	ER Sepsis Triage	A
H	2014-03-11 10:22:00+01:00	CRP	B
H	2014-03-11 10:22:00+01:00	LacticAcid	B
H	2014-03-11 10:22:00+01:00	Leucocytes	B
H	2014-03-11 13:54:26+01:00	Admission IC	J
H	2014-03-12 07:00:00+01:00	Admission NC	J
H	2014-03-13 07:00:00+01:00	CRP	B
H	2014-03-15 07:00:00+01:00	Leucocytes	B
H	2014-03-15 07:00:00+01:00	CRP	B
H	2014-03-15 11:36:15+01:00	Admission NC	K
H	2014-03-16 01:00:00+01:00	Release B	E
...	...	...	...

- Comparative Process Mining: comparing multiple event logs and extracting information such as similarities and differences, and, in this way, understanding the reasons for differences.

- Predictive Process Mining: using machine learning to make a prediction given that processes of organizations are subject to changes.

- Action-Oriented Process Mining: turns diagnostics into actions, such as adding resources when waiting times exceed a threshold or notifying the manager when a task is skipped by an employee.

2.3 LLMs

LLMs are sophisticated AI models trained on massive amounts of text data, enabling them to understand, generate, and translate human language[5]. Built upon deep learning architectures, particularly transformers [12], LLMs can perform a wide range of tasks, including generating human-like text, translating languages, writing different kinds of creative content, and answering questions in an informative way. To better understand such powerful tools, each of the following Sections presents the building blocks that led to their existence.

[5] https://www.ibm.com/think/topics/large-language-models

2.3.1 Statistical Language Models (SLMs)

As presented in Figure 2.3, in the 1990s SLMs were introduced [2]. Those models use probabilistic methods to analyze contextually relevant aspects of natural language, estimating the likelihood of a sentence occurring within a context. Using conditional probability, the probability of each word is calculated based on the previous words in a sentence and the predicted word is the one with highest probability. These methods typically assume that each token depends on the previous n tokens in the sequence and this is referred to as n-grams [18].

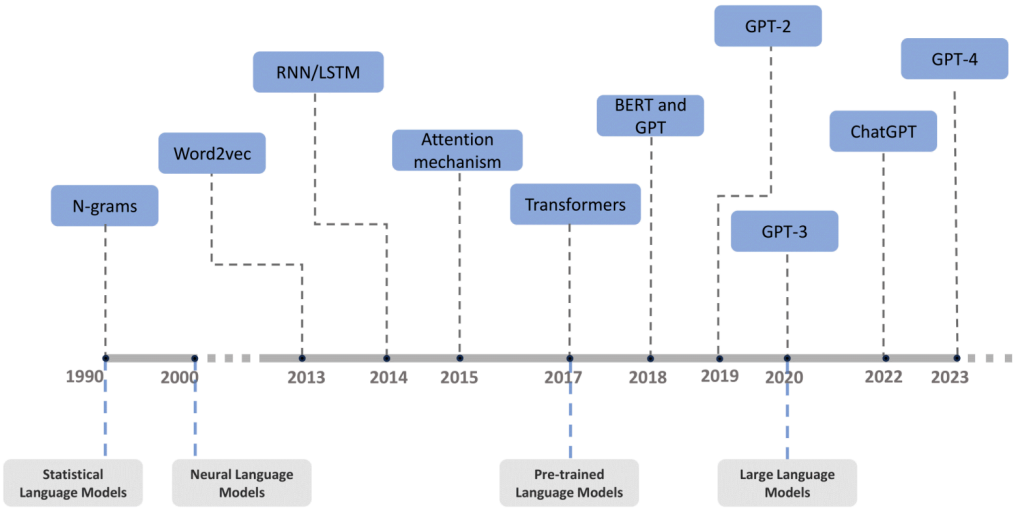

Figure 2.3: LLM History. Source: [2]

However, SLMs and n-grams can only consider local context (i.e., limited by the size of n which was typically small). Additionally, even large corpora are often insufficient to cover all possible n-length sequences, and it is harder for them to deal with rare/unseen words. A lot of these issues were addressed by the evolution of neural networks.

2.3.2 Neural Networks (NNs)

NNs are inspired in the biological neuron of the human brain which receives input signals and sends output signals to other neurons through connections called synapses [3]. To replicate that behavior, NNs are composed of artificial neurons, where each of them receives a pre-activated input and then processes it with an activation function:

- Pre-Activation: it is performed by Equation (2.1), where w is the weight of the connections, determining the importance of each input, and b is a bias, which shifts the input and gives more flexibility.

$$z(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{D} w_i x_i + b \tag{2.1}$$

- Activation: the main goal of activation functions corresponds to the addition of non-linearity which allows NNs to learn complex patterns and relationships. Different functions are suited to specific tasks; for instance, sigmoid functions are good for probabilistic outputs while softmax functions output normalized probabilities and are often used in the output layer for classification tasks.

The artificial neurons can be grouped in multiple layers. This way creating a NN (Figure 2.4). The simplest NNs are referred to as Feed Forward Neural Networks (FFNNs)/Multi-Layer Perceptrons (MLPs) and they contain the following fundamental layers:

- Input Layer: receives input data.
- Hidden Layers: perform computations. Different hidden layers can look at the input differently and recognize different features. This means that adding hidden layers increases the expressivity of the NN.
- Output Layer: delivers output data.

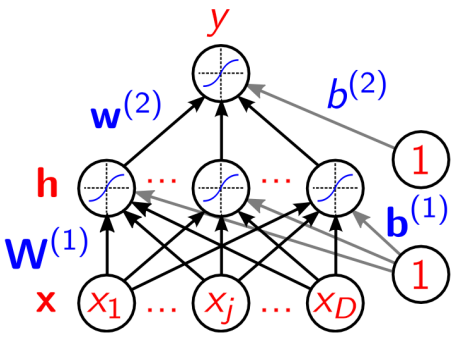

Figure 2.4: Neural Network Example. Source: [3]

NNs can approximate functions, and this capability forms the basis of their training process. In order to learn, NNs update their weights and biases by relying on data. To guide these updates, a loss function measures the network's error; and using backpropagation, the network figures out how each weight and bias contributed to the error. The loss function used during training is defined as follows:

$$\mathcal{L}(\boldsymbol{\theta}) := \sum_{i=1}^{N} L\left(\boldsymbol{f}\left(\boldsymbol{x}_i; \boldsymbol{\theta}\right), y_i\right) \tag{2.2}$$

where:

- N is the total number of training examples.
- $\boldsymbol{x}_i$ represents the i-th input example, and y_i is its corresponding true label.
- $\boldsymbol{\theta}$ denotes the parameters (weights and biases) of the neural network.
- $\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta})$ is the predicted output of the neural network for input $\boldsymbol{x}_i$.
- $L\left(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i\right)$ is the loss function that measures the difference between the predicted output and the true label.

The objective of training is to minimize $\mathcal{L}(\boldsymbol{\theta})$ by iteratively updating the parameters $\boldsymbol{\theta}$. This process ensures that the model learns to make predictions that closely align with the true labels.

2.3.3 Recurrent Neural Networks (RNNs)

RNN is a type of NN that allows us to deal with sequential data with an arbitrary length, which is quite common (e.g., words in a sentence, DNA sequences). Unlike traditional FFNNs where information flows only in one direction, RNNs have connections that loop back on themselves, allowing them to "remember" past information without being constrained by length. As we unroll them, they can be visualized as represented in Figure 2.5.

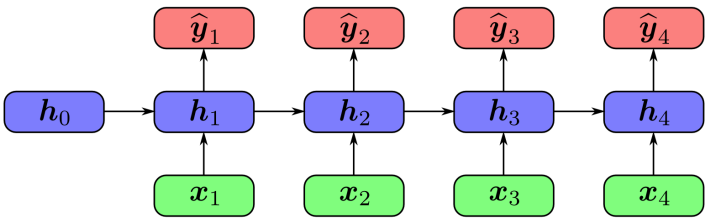

Figure 2.5: Recurrent Neural Network. Source: [3]

More formally, RNNs handle sequential data by taking as input at each time step a representation of the current token/sequence unit x_t and combining it with its previous hidden state h_{t-1} to compute the current hidden state h_t (Equation (2.3)). This way, each hidden state h_t acts as a compressed representation of the sequence's history up to point t. In Equation (2.3), g represents an activation function; V and U represent weight matrices; and c is the bias.

$$h_t = g(Vx_t + Uh_{t-1} + c) \tag{2.3}$$

Equation (2.4) shows how to calculate the current output $\hat{y}_t$ based on the current hidden state h_t.

$$\hat{y}_t = Wh_t + b \tag{2.4}$$

RNNs, while effective for sequential data, face challenges when processing long sequences. During backpropagation, the repeated multiplication of gradients can lead to very small (vanishing) or very large (exploding) gradients, hindering the learning process. To address these limitations, gating mechanisms were introduced, leading to the development of more sophisticated architectures such as Gated Recurrent Unitss (GRUs) and Long Short-Term Memorys (LSTMs).

2.3.3.A LSTM

LSTMs [19] are a type of RNN designed to better handle long sequences of data and learn long-term dependencies. LSTMs have introduced a special structure called memory cells (Figure 2.6) that allow them to remember or forget information over long time periods and in order to control that flow of information, they use a gating mechanism. Specifically, an LSTM cell has the following gates:

- input gate: decides what relevant information shall be added to the memory cell;
- forget gate: decides which information can be ignored; and
- output gate: decides how much information should be used to compute the output.

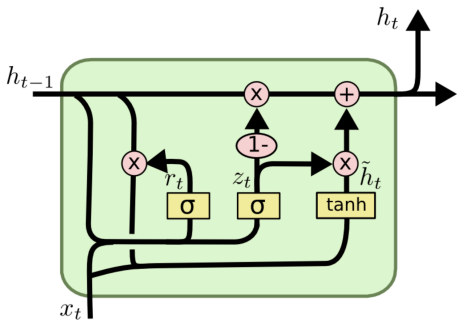

Figure 2.6: LSTM's Memory Cell. Source: Christopher Olah

2.3.4 Transformer

While designed to handle long-term dependencies, LSTMs still struggle to capture very long-range information due to their sequential nature. To address that, a Transformer [12] uses an attention mechanism that allows it to capture those long-range dependencies more effectively and simultaneously. This way, it can translate an input sequence into an output sequence. To better contextualise, a description of attention, as well as the encoder and decoder blocks of a typical transformer model (Figure 2.8) are provided.

2.3.4.A Self-Attention

Self-Attention is a mechanism that allows a model to weigh the importance of different parts of the input sequence when processing a specific position, enabling it to understand the relationships between words and capture the overall context of the input (Figure 2.7(a)). It can weigh the importance of words without being constrained by the word order in the sentence.

(a) Encoder Self-Attention. (b) Decoder Self-Attention (masked).

Figure 2.7: Self-Attention. Source: [3]

To calculate the Self-Attention, first it is necessary to calculate three sets of values: Queries (Equation (2.5)), Keys (Equation (2.6)) and Values (Equation (2.5)). Those values are all calculated using input sequence X and different weight matrices.

$$Q = XW^Q \tag{2.5}$$

$$K = XW^K \tag{2.6}$$

$$V = XW^V \tag{2.7}$$

As presented in Equation (2.8), Q_i and K_j values are used to calculate a similarity-like type of score of word i with the rest of words j (note that the highest similarity happens when $i = j$, since the word is similar to itself).

$$\text{Score}(Q_i, K_j) = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}} \tag{2.8}$$

To give the proper weight of each word on the new encoding of the current word, the score previously calculated, goes through a softmax function. This determines the proportion allocated to each word before it is multiplied by V (Equation (2.9)).

$$\text{Attention}(Q, K, V) = \text{softmax}(Score(Q, K)) \cdot V \tag{2.9}$$

We can repeat that process with multiple heads, each of them with different W^Q, W^K and W^V to

capture different relationships between words. That is called Multi-Head Attention.

2.3.4.B Encoder & Decoder Blocks

The first step in an encoder block (Figure 2.8) is to obtain an input representation (e.g., using the word embedding generated for each word by Word2Vec [20]) and the respective positional encoding, which stores information about the position of each word in a sentence. Subsequently, the input is forwarded to a Self-Attention Layer. The Encoder Block finishes with a FFNN where each input is processed independently. This not only helps learning more complex features but contributes to efficency.

As presented on Figure 2.8, the first component of the decoder corresponds to the Self-Attention Layer that will maintain the relationship among words in the output, but since words in the output still need to be generated, there needs to be a "masking". Masked Self-Attention only calculates the similarity of the current word with itself and with the previous words, having an autoregressive nature (Figure 2.7(b)). The following layer corresponds to Encoder-Decoder Attention which decides how much importance should be given to each word of the input when generating the future word of the output, this is done by calculating the similarities of the current output Q value and input K values. The Decoder Block finishes with a FFNN that maps the result of Encoder-Decoder Attention to a token in the output vocabulary.

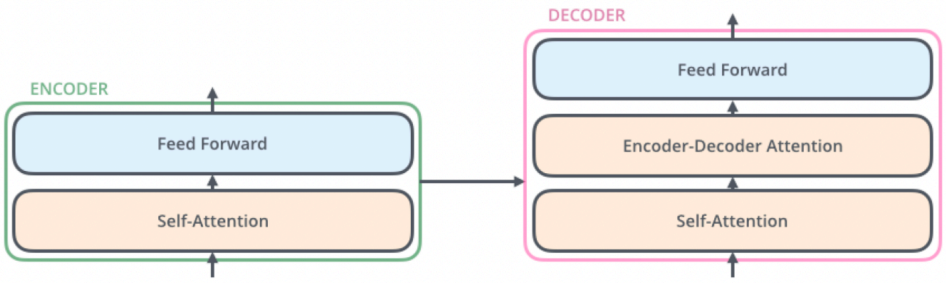

Figure 2.8: Transformer Block. Source: [3]

The Encoder Block is responsible for processing the input context by extracting meaningful information from it and storing that information on what is called a context vector [21]. Encoder-only architecture is the building block of models like BERT [22]. On the other hand, the Decoder is responsible for generating the output in an autoregressive manner and given its usefulness in machine translation and dialogue, Decoder-only is the basis for models like GPT-3 [23].

2.3.5 Pre-trained Language Models (PLMs)

In 2017, Transformers, which are described in Section 2.3.4, were introduced and they are the foundation of PLMs. PLMs follow a two-phase learning paradigm: pre-training and fine-tuning.

During pre-training, a model learns foundational language structures—such as vocabulary, syntax, and semantics—by training on extensive unlabeled text corpora. This is typically done via causal language modeling (next-token prediction).

In the fine-tuning phase, the pre-trained model is adapted to perform well on a specific, often smaller, dataset. This process, a form of transfer learning, leverages the knowledge learned from the original training data to improve performance on a specific task. Fine-tuning involves selecting a suitable pre-trained model, preparing the target dataset, and iteratively adjusting the model's parameters on the new data to achieve optimal performance[6].

2.3.6 LLMs

With the availability of large amounts of diverse data, the algorithmic innovation previously mentioned and the increased computational power made possible by companies like Nvidia[7], it is now possible to train LLMs on massive text corpora using billions of parameters. For instance, in 2024, Meta introduced Llama 3 with 8 and 70 billion parameter scale[8]. That open-source LLM has great capabilities in tasks associated with reasoning, code generation and others.

2.3.7 Advanced Topics

2.3.7.A Prompt Engineering

Prompt Engineering corresponds to understanding how to interact with LLMs. By carefully structuring the input, users can influence the LLM's behavior in order to obtain the expected outcome. Some approaches that can be followed and that lead to improved responses, correspond to[9]:

- Writing the instruction clearly (e.g., if someone wants a specific output format, they should specify it [10]).
- Specify the desired length of the output.
- Asking for the reference to guarantee that the information is reliable.
- Instead of only giving the LLM an instruction, provide it with a few examples that can help understand a pattern (further described below).

In-context learning, which underlies some of these strategies, refers to the ability of LLMs to learn from examples presented directly within the prompt—without any parameter updates. Techniques such

[6] https://learn.microsoft.com/en-us/windows/ai/fine-tuning
[7] https://www.nvidia.com/en-eu/
[8] https://ai.meta.com/blog/meta-llama-3/
[9] https://platform.openai.com/docs/guides/prompt-engineering
[10] https://medium.com/@mengsaylms/mastering-prompt-engineering-for-effective-llm-output-tips-techniques-and-warning-d76b09515c3

as zero-shot, one-shot, and few-shot prompting exploit this capability to elicit strong performance by leveraging the model's pre-trained knowledge through carefully curated instructions and demonstrations. While zero-shot prompting relies solely on task instructions without examples, one-shot and few-shot variants provide one or multiple demonstrations to guide output format and reasoning [24].

2.3.7.B RAG-LLM

LLMs, once trained on a specific dataset, acquire static knowledge, not being able to account for updated information. This, in turn, can lead a model to generate hallucinations (inaccurate responses), making information retrieval useful in certain cases. RAG-LLM can access external databases like document repositories (e.g., Wikipedia) and search engines (e.g., Bing) to complement the standard model [25].

More formally, and as can be seen in Figure 3.2, conditioned on the input, k documents are extracted by a retriever and then, conditioned on the input and the retrieved documents, the output is generated [5]. In Section 3.4, a more detailed overview of this approach is presented as well as its great results.

2.3.7.C CoT

CoT mimics human reasoning by breaking problems down into multiple sequential steps in order to solve them, mirroring the cognitive strategy of breaking down complex tasks into manageable, intermediate thoughts[11]. CoT Prompting consists of presenting some demonstrations of CoT to a given model in order to have the model present a similar one. This can be used with arbitrary LLMs and have them accurately solve specific tasks such as arithmetic problems and common sense reasoning [26]. In Section 3.5, the capabilities of CoT in those tasks and others is presented.

2.4 Ambiguity

2.4.1 Ambiguity Description

The following overview is based on the entry on ambiguity from the Stanford Encyclopedia of Philosophy [27].

Ambiguity is a term hard to define but tends to refer to the capacity of signs, such as words and symbols, to possess multiple valid interpretations within a given system. From a philosophical perspective, the interest arises in natural language, given its inherent flexibility and potential for multiple interpretations which pose a significant challenge for formal systems that aim to represent meaning precisely. Aristotle referred to it as either the situation where a word/expression has multiple meanings or is used by custom, or when words have a clear meaning when used separately, but can have multiple interpretations when used together.

[11] https://www.ibm.com/think/topics/chain-of-thoughts

We can detect different types of ambiguity. For instance, they can be lexical (homophonous words or even co-spelled words can have two different meanings - ”bat” can refer to either the animal or the object), syntactical (a sentence can have different syntactical structures and consequent different interpretations - ”superfluous hair remover” can either mean ”hair remover that is superfluous” or ”remover of hair that is superfluous”), semantical (the same sentence can have two different meanings - ”The chicken is ready to eat” can mean the chicken is ready to be fed or to be fed to someone), or even associated with speech acts (utterances can perform different actions beyond simply conveying information - ”the cops are coming” can be a warning or an assertion).

Furthermore, ambiguity has significant implications in various domains. In law, ambiguous legislation can undermine its enforceability. On the other hand, in arts, it can enrich artistic expression by fostering multiple interpretations and inviting deeper engagement with the work.

2.4.2 Ambiguity in NLP

Considering LLMs, mechanisms like self-attention can be useful in the disambiguation of words that have different meanings. For instance, words like “bat” have different meanings as previously mentioned, but since self-attention encodes the word considering its context, it will be able to understand which of the meanings is more relevant in that type of case. However, there are more specific situations where LLMs might not initially have enough information to disambiguate. In the paper introduced by Ginzburg [28], two types of ambiguity are highlighted: coreference resolution, which involves identifying what a pronoun refers to in a sentence; and answer types, which pertain to the uncertainty regarding the nature of the desired answer (e.g., when asked about a book, not sure if it is about identifying the title or genre of it).

Even before LLMs, ambiguity has been prevalent in natural language requirements for software development, which can cause misinterpretations among stakeholders. This context is usually associated with pragmatic ambiguities which arise from the reader’s background knowledge, which varies widely and cannot be addressed solely with rule-based tools. Multiple works have been developed to address those kinds of ambiguities using knowledge graphs and other methodologies as described in 3.3.

2.4.3 Ambiguity in Process Mining

Event logs can be affected by several data quality issues that complicate process analysis. These include missing data, which creates ambiguity when essential information like a case ID or timestamp is absent; multiplied data, where duplicate entries of the same event occur with slight attribute variations; and incorrect information, where the recorded execution is misaligned with the actual process model. Furthermore, imprecise information, such as a timestamp only indicating date and not time, can obscure the true sequence of activities. Finally, unexpected information in the log may deviate so significantly from expected values that interpreting the execution against a discovered model becomes challenging [29]. All

those quality issues introduce ambiguity in the field of process mining.

Beyond these general data quality problems, ambiguity also arises from the specific design of the log. If activity names are too coarse and certain activities consistently occur in immediate sequence, it becomes difficult to discern their logical relationship, introducing another layer of uncertainty during analysis [30].

The ambiguity present in natural language also affects this field in a different way. As outlined in the previous section, pragmatic ambiguity can lead to misunderstandings. For example, a user querying an event log might ask, ”What is the diagnosis after Triage?” The term ”diagnosis” has entirely different meanings in the context of a sepsis case log versus the context of process mining. In process mining, the ideal answer would be the next possible activity, whereas in the sepsis context it would be a medical conclusion like ”has sepsis” or ”doesn’t have sepsis”.

3

Related Work

Contents

This chapter situates our research and implemented system within the landscape of applying LLMs to process mining tasks (Section 3.1). A key challenge we address is the lack of process mining datasets tailored for NLP tasks; consequently, we review how previous work has utilized different data abstractions, which informed the creation of our own dataset (Section 3.2). Furthermore, given the lack of existing approaches to handle natural language ambiguity in process mining conversational systems, we outline prior methods for dealing with ambiguity in software engineering and their evolution within LLMs (Section 3.3). To ground our system's responses, we present the use of RAG-LLM to complement a model's parametric memory, a technique we employ for retrieving event logs from a repository (Section 3.4). Finally, we conclude by presenting the capabilities of CoT reasoning, which motivated our novel approach to resolving ambiguous queries (Section 3.5).

3.1 LLMs & Process Mining

In the context of business processes, significant efforts have been dedicated towards generating process models from textual descriptions, such as ProMoAI [31] and BPMN-ChatBot [32].

Considering the field of process mining, the work of Rebmann et al. [33] proposed in 2024, focuses on the utility of LLMs for solving tasks based on their general process knowledge and the semantics of activities and their interrelation. The tasks include (a) trace-level semantic anomaly detection (T-SAD) - a binary classification task where a process trace is classified as anomalous based on its semantics; (b) activity-level semantic anomaly detection (A-SAD) - evaluating pairwise activity relations within traces to classify them as valid or anomalous; and (c) semantic next activity prediction (S-NAP) - predicting the next activity in a process while incorporating semantic understanding. Two approaches were followed - fine-tuning and Few-Shot In-Context Learning - and the results showed that fine-tuning was the most effective strategy for the resolution of the tasks. Furthermore, activity-level semantic anomaly detection (A-SAD) was the easiest task and LLMs demonstrated strong capabilities, particularly for specialized processes where their general process knowledge proved valuable. However, they occasionally misclassified valid traces as anomalous when additional context was needed, highlighting a shared challenge between models and humans. This work was targeting specific tasks.

Another work that presents an approach capable of solving a wider range of tasks and further bridges the gap between the usefulness of process mining in different sectors and the required process mining knowledge from the user to extract information from processes, is C-4PM [4]. C-4PM is presented as a conversational agent grounded on reasoning and logical inference, capable of providing declarative process mining models to non-expert users.

Based on the Rasa Framework, and as can be seen in Figure 3.1, C-4PM has two main components, the Rasa Stack and the Action Server. Within the Rasa Stack, the Rasa NLU is the module responsible for processing the input. The module performs three tasks: (a) Intent Recognition - extraction of the objective of the user, which can be one of nine (e.g. specification description, list activities); (b) Entity Recognition - extraction of relevant entities of a process; and (c) Entity Linking - association of the identified entities to the corresponding elements of the event log or business process. The element within the Rasa Stack responsible for dialogue management (i.e., defining next action), is the Rasa Core. Moving to the Action Server, the NL2LTLf is used in specific cases to transform natural language input into a LTLf formula [16] compatible with Declare4Py using the help of GPT model. Declare4Py, grounded on the DECLARE language, is a python library that uses constraint-based specifications to analyze a process. C-4PM uses it to find the declarative specification of a process and take care of the reasoning component according to the initial intent and entities identified. Having a template for each intent, a response is generated following a semantic and syntactic structure. A layer with GPT-3 [23] further refines the answer according to the context and the question initially presented, presenting more fluidity

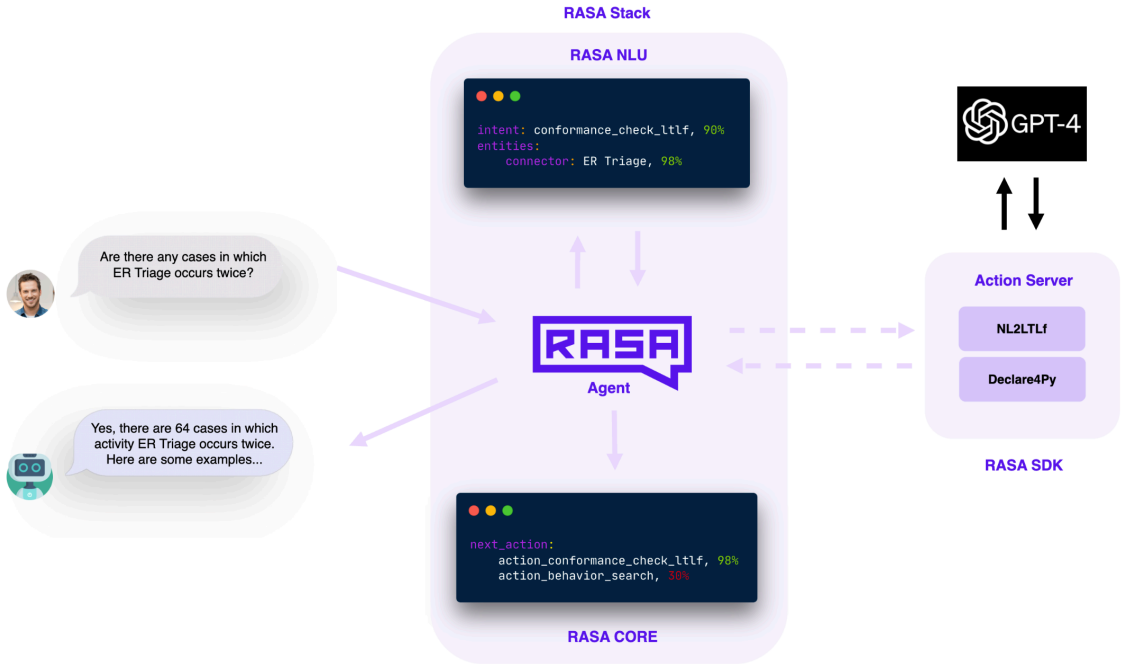

Figure 3.1: Architecture of C-4PM. Source: [4]

and other human-like characteristics.

In summary, we have seen how LLMs are being used to solve process mining tasks and how successful they are in tasks like anomaly detection. At the same time we have seen the architecture of conversational agents for process mining like C-4PM [4] that delegate the task to Declare4Py. However none of them deal with the inherent ambiguities of natural language and none of them resort to CoT reasoning.

3.2 Datasets

Considering the multiple representations for a given business process, multiple abstractions can be considered in order to have a dataset that maximizes the extraction of valuable insights. Aimed at improving interpretability, Kourani et al. [34] presented multiple methods to abstract the business process and evaluate which abstraction yields better results: XML code, which encapsulates the visual and structural components of BPMN model; simplified XML, which eliminates non-relevant information from XML code, such as styling; JSON which maintains only the essential information on its list of elements, each of them containing a set of attributes; and images (PNG). Their results showed that different abstractions were appropriate for different tasks but, in general, simplified XML and JSON presented better results while consuming less tokens when compared to PNG and standard XML.

Specific to process mining, the event log could be considered as context for the conversational agent

to solve a given user query. However, large event logs make it challenging for LLMs to extract valuable information from them. With that in mind, Berti et al. [35] studied the use of different event log abstractions that reduce the constraints in size. Those abstractions include process flow (abstraction of DFGs) and process variants; and correspond to the abstractions that we are adopting in the present research.

3.3 Dealing with Ambiguity

Ambiguity has long been a challenge in computer science, even before LLMs. One example is the work of Ferrari et al. [36], which addressed pragmatic ambiguity (described in Section 2.4.2) in natural language requirements. Their method creates artificial "subjects" with different backgrounds, modeled as domain knowledge graphs built from different domain-specific documents. Nodes representing key concepts (stems of relevant terms), and edges capturing co-occurrences with direction and weight. Each requirement is interpreted by traversing these graphs, activating concepts that form a pragmatic interpretation. By comparing interpretations across multiple knowledge graphs (different graphs will have different activated concepts for the same requirement), the method detects ambiguity: high similarity implies clarity, while divergence suggests ambiguity.

Another approach that addresses the problem of ambiguity within the context of software engineering is the one presented by Ferrari and Esuli [37], which specifically deals with identification of terms that may lead to misunderstandings in requirements elicitation scenarios involving stakeholders from different domains. The approach builds domain-specific language models for each domain, using word embeddings to identify semantic variations between domains. By measuring the variations in usage with the help of word embeddings, the method estimates the ambiguity potential of each term.

With the introduction of LLMs, the issue of ambiguity gained even more importance considering that those tools have to deal with the ambiguities that arise in conversation scenarios. Aimed at solving ambiguities in conversations based on questions and answers, Abg-CoQA [38] focuses on identifying ambiguities (Ambiguity Detection), following up with clarification questions (Clarification Question Generation) - questions whose answers will eliminate ambiguity and allow a given model to answer the initial question that was previously ambiguous - and returning a final answer (Clarification-based Question Answering). The results of Abg-CoQA [38], highlight how challenging it was to identify ambiguity, as well as generate clarification questions and respective answers since all models used in the experiment lagged behind human performance.

In summary, previous work highlights how relevant and challenging it is to deal with ambiguity. Through our system, one of the ambiguities we address corresponds to pragmatic ambiguity. We also try to identify specific terms that might introduce ambiguity and, instead of asking a clarification question, our system assumes the most probable non-ambiguous query based on the identification of ambiguous

terms.

3.4 Retrieval Based Models

As mentioned in Section 2.3.7.B, RAG-LLMs represent an approach that can help deal with hallucinations and improve reliability when providing an answer. A study of relevance that highlights RAG-LLM's advantages and components was conducted by Lewis et al. [5] in 2020. The RAG-LLM in that research employed a parametric memory (a seq2seq model [39]) alongside a non-parametric memory (dense vector index of Wikipedia), where the dense vector index was accessed with a retriever - Dense Passage Retriever [40]. Focusing on the non-parametric memory, for each input, the model retrieved k relevant documents and generated output conditioned on both the input and retrieved documents (Figure 3.2). For the experiments, a single Wikipedia dump was used for the non-parametric memory, and RAG-LLM was tested on multiple knowledge-intensive tasks such as open-domain question answering, jeopardy question generation, and fact verification. Depending on the task, RAG-LLM either achieved state of the art results or surpassed them (e.g., in open domain QA). Its non-parametric memory not only allowed for easy updates but also enabled it to produce more diverse outputs than models like BART [39].

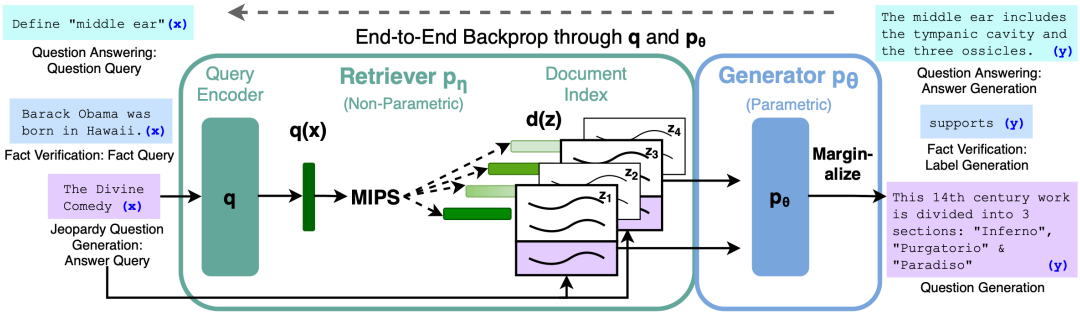

Figure 3.2: RAG-LLM approach For NLP Tasks. Source: [5]

While that work showed promising results on the use of RAG-LLM for NLP tasks, the ambiguity that might exist in a given query and the indiscriminate context use were challenges that were not considered. To address those specific challenges, RQ-RAG [25] was introduced. RQ-RAG refines queries for retrieval, leveraging Llama2 [41] and enhanced with ChatGPT to rewrite, decompose, and clarify ambiguities. This refinement enables the model to identify the intent of the query before initiating information retrieval, providing a more effective search strategy.

In this thesis, we start by seeing the performance of the model we selected for answer generation without considering context retrieved from external repositories. Then, motivated by the work presented in this section, we introduce a naive approach to RAG-LLM that allows our system to retrieve an event log from an online repository.

3.5 CoT Reasoning

As mentioned in Section 2.3.7.C, CoT solves a problem by breaking it into sub-problems and this is useful in the mathematical context. For instance, the application of CoT reasoning with bounded-depth autoregressive transformers to solve mathematical and dynamic programming problems is explored in [42]. For arithmetic problems, the researchers demonstrated how CoT could decompose expressions into smaller components, solve them individually, and then combine the results to reach a final solution. In the case of linear equations, the model iteratively eliminates variables in equations to compute their values through back-substitution, replicating the Gaussian elimination algorithm. For dynamic programming, a method where problems are addressed by being divided into subproblems and being solved sequentially, the proposed transformer architecture included components to calculate problem size, implement transition functions, calculate and aggregate subproblem results to compute final solutions. Analysing the results from that work, models trained on CoT datasets (<problem, CoT steps, answer>) achieved near-perfect accuracy on arithmetic expressions with 4–6 operators, linear equations with 3–5 variables, Longest Increasing Subsequence with lengths of 50, 80 and 100; and Edit Distance with string lengths of 12, 16 and 20. Furthermore, as input size increased, CoT models outperformed Direct models, which showed a notable decline in accuracy.

CoT prompting is a method that generates natural language rationales to guide problem-solving and leverages few-shot learning. The work of [26] explores enhancing the reasoning capabilities of large language models using this technique. Empirical evaluations of this study demonstrate CoT's effectiveness in benchmarks like GSM8K [43], CSQA [44], and symbolic tasks. For arithmetic reasoning, CoT significantly improves performance in large-scale models ($\geq 100B$ parameters), where gains are most pronounced on complex problems. Smaller models, however, often produce incorrect reasoning steps. In common sense reasoning, [26] shows that CoT continues to demonstrate gains with larger models, improving tasks like date and sport understanding. CoT prompting using PaLM 540B achieved nearly 100 % success rates on symbolic reasoning tasks, particularly in the in-domain evaluations, where the tasks were straightforward. However, smaller models also struggled with abstract manipulations.

The approaches mentioned above illustrate that CoT excels in solving complex problems requiring structured step-by-step reasoning, such as arithmetic problems. The second work also evaluates CoT performance on common sense reasoning. Building on these insights, we hypothesize that CoT prompting can be used to disambiguate a user query before executing a process mining task. The reasoning steps provided by CoT would guide the model from an ambiguous input to a refined query and an actionable response.

4

Proposed System

Contents

This chapter starts with a high-level overview of the implemented system in Section 4.1. This is done by describing the sequence of events and tasks performed from the moment the user provides a query to the system up until they receive an answer.

The design of our system leads to a constant enhancement of the prompt, which led to the production of multiple datasets as the development and implementation of the pipeline progressed. In Section 4.2, the different components of the system are described in detail, following the sequence by which they were added: starting by the creation of the initial dataset, implementation of naive retriever, progressing to the creation of the ambiguous dataset, mentioning the CoT implementation and finalizing by listing all the datasets that were produced as a result of progressing through the pipeline. All code and data used in our experiments is available at https://github.com/lucas-neves-23/Amb-PM-LLM-Pipeline.

4.1 System Overview

As highlighted in Chapter 3, there have been multiple efforts to provide conversational agents that can solve process mining tasks; at the same time, we have seen how relevant the challenge of ambiguity is when dealing with LLMs. The implemented system, represented in Figure 4.1, incorporates our approach to RAG-LLM - specific for event log retrieval - and CoT - for query disambiguation - and aims to build on top of both dimensions to be capable of solving process mining tasks while dealing with the inherent ambiguity of natural language that hinders LLMs from providing more robust answers.

To do that, once the user provides a query related to a given event log, the system performs the following actions:

1. RAG: Identify Event Log - starts by identifying the specific log that the query provided by the user is referring to. This step employs lightweight keyword matching against an Online Repository of publicly available event logs to select the most relevant one.

2. Retrieve Event Log - the respective event log file is retrieved from the Online Repository.

3. Obtain Abstraction of the Event Log - from the retrieved event log, a compact abstraction consisting of Process Flow(DFG relations) and Process Variants is produced. That abstraction is added to the user prompt. This ensures that subsequent reasoning is anchored in factual process structures rather than parametric knowledge alone.

4. CoT steps - CoT reasoning steps consisting of Ambiguity Identification, followed by Ambiguity Association, suggestion of Non-Ambiguous Query and respective answer. Those steps are added to the new user prompt with the goal of assisting in query disambiguation.

The final prompt provided to the model before answer generation thus incorporates the retrieved process flow and variants as context (contributing to objective O2 of this thesis), followed by the original user query and explicit CoT instructions to guide ambiguity resolution and response generation (contributing to objective O3).

4.2 Implementation

For the implementation of the system previously outlined, the first decision that was made consists of the event log that would be used for the creation of datasets, running of experiments and exemplification of the system's behavior. Since there is a lot of previous work associated with the Sepsis Cases - Event Log[1], that was the one we proceeded with. The Sepsis Cases - Event Log is a publicly available dataset used for process mining research, containing real-life patient data from a hospital's Enterprise Resource

[1] https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639

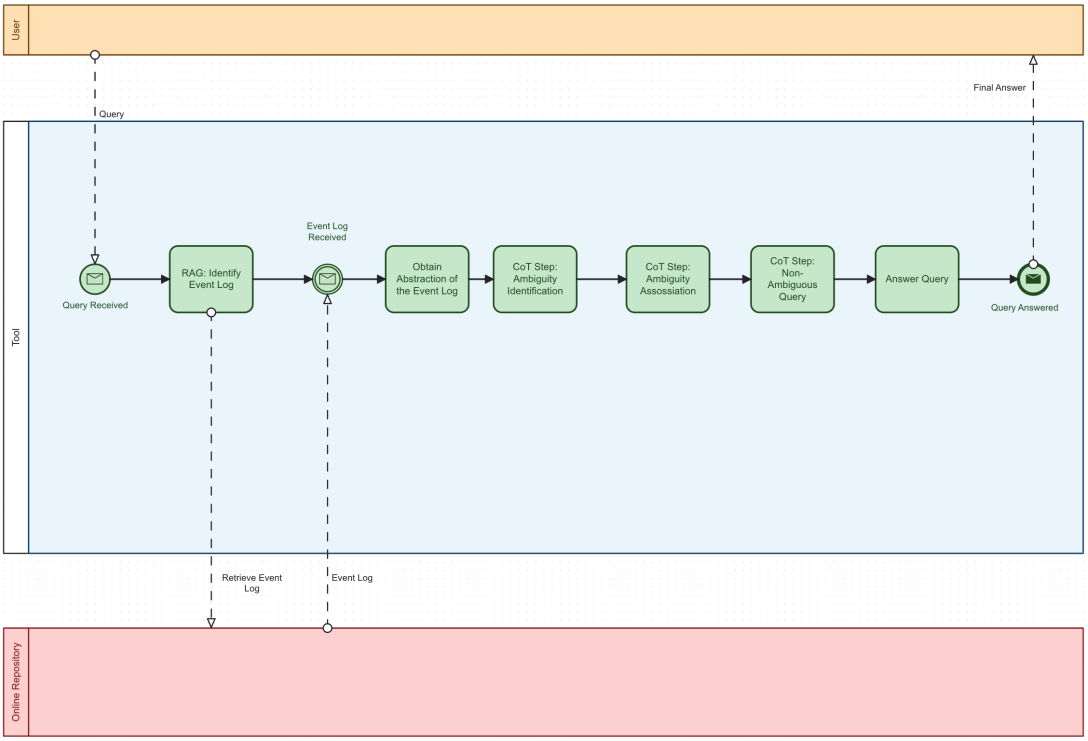

Figure 4.1: System Overview.

Planning (ERP) system. It contains 1,000 cases with in a total of 15,000 events; 16 different activities; and attributes related to the logistical and treatment pathways of patients suspected of having sepsis, providing insights for understanding and improving patient care by identifying bottlenecks and deviations from guidelines. A small fragment of that event log is presented on Table 2.1.

This decision contributed to the production of question-answer pairs as described in the following Section, as well as, to the adoption of an abstraction which is necessary given the extreme size of the event log.

4.2.1 Initial Dataset

For the creation of the initial dataset comprised of question-answer pairs, we resorted to the C-4PM [4] data available on github which included an extensive set of questions related to the sepsis process[2]. Having those questions, we relied on pm4py[3] - a process mining library for python - for the analysis of the event log which allowed us to obtain concrete solutions for the considered questions. From the concrete solutions, the answers were formulated, leading to a dataset comprised of 8 categories of process mining tasks. Table 4.1 outlines the distribution of the resulting dataset, highlighting the number of instances/samples related to each specific task.

[2] https://github.com/Yagouus/c-4pm/blob/main/data/nlu.yml
[3] https://processintelligence.solutions/pm4py

Table 4.1: Initial Dataset Distribution Over Different Categories of Tasks

Category	Description	Number of Instances
C1	List the different activities	7
C2	Describe the underlying process model	14
C3	Provide existing conformant trace examples	7
C4	Generate new conformant trace examples	5
C5	Confirm the existence of an underlying process model (Yes/No)	21
C6	Behavior check (Yes/No regarding 'followed by' relation, etc)	20
C7	List possible next activities	3
C8	Confirm if a given trace is conformant with the model (Yes/No)	10

4.2.2 Dataset Enhanced with Naive Retriever

Considering the problem of ambiguity during context retrieval as highlighted in Section 3.4 and tackled by [25], we opted for a computationally efficient approach that would not require query disambiguation before the retrieval, leaving the disambiguation task for the CoT component. This way, during the reasoning steps, the event log abstraction would already play a role in disambiguating the user query. With that in mind, the approach we considered corresponds to word overlap for the identification of the event log that is being referenced in a given question. This requires that the event log be referenced by the user whenever asking a question. Note that this still allows for a user-friendly interaction since, while the user might lack the formal knowledge that introduces pragmatic ambiguity, they always know what the domain and event log they are working with are.

The retrieval component, implemented in python, accesses the open repository of eXtensible Event Stream (XES)-formatted event logs hosted by the 4TU.ResearchData platform[4]. This access is done via dynamic scraping from the Task Force on Process Mining index[5].

Considering Figure 4.2, the formal sequence of steps for context retrieval corresponds to:

1. The retrieval begins by issuing a `HTTP GET` request to the fixed endpoint of the Task Force on Process Mining index and parses the response to extract a dictionary of available logs. Each entry maps a descriptive log name (e.g., "Sepsis Cases - Event Log") to its corresponding DOI URL.

2. Given a user query, the module then selects the best-matching log via lexical overlap. It starts by checking if there is any overlap between the query and any log name obtained from step 1, prioritizing the log with the maximum overlap. With this implementation, if a user writes "sepsis" or "Sepsis Cases" in a given query, the identified event log will be "Sepsis Cases - Event Log". On the other hand, if the user writes "Conformance Checking Challenge", "Conformance Checking Challenge 2019", or "CCC19", the identified event log will be "Conformance Checking Challenge 2019 (CCC19)". There is a fallback consisting of partial matching - used in case there are any typos

[4] https://data.4tu.nl/
[5] https://www.tf-pm.org/resources/xes-standard/about-xes/event-logs

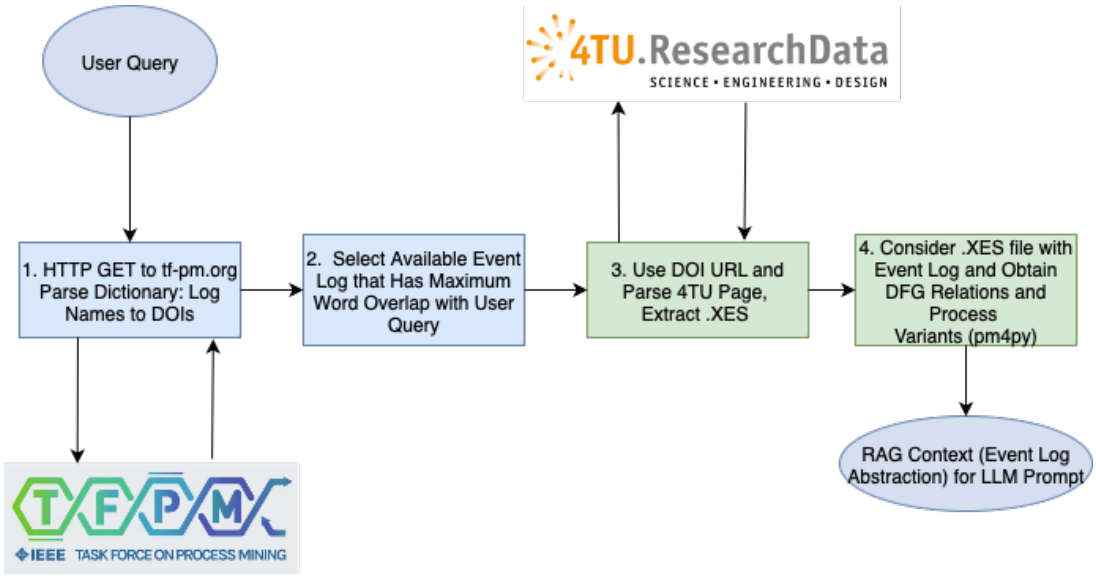

Figure 4.2: Retrieving and Abstracting Event Log.

or abbreviations.

3. Upon matching with a given event log name, the module uses the respective DOI and fetches the log file. This is done by parsing the DOI-resolved 4TU landing page to locate the download link.

4. The final .xes file obtained from the download link is copied to the working directory. Then the process abstraction is obtained using pm4py. As presented in Listing 4.1, this abstraction consists of DFG relations/process flow and process variants and its content permits the resolution of tasks belonging to the categories in Table 4.1. This abstraction is of extreme relevance and while we could have considered implementing a more refined RAG-LLM and dividing the raw XES event log file into chunks for retrieval, we would be losing information since some process mining tasks like conformance checking require knowledge of the whole event log in order to perform an effective analysis. Note that the model used during the experiments - Meta-Llama-3-8B-Instruct[6] - has a context length of 8k which is not enough to consider the raw XES event log file. The adopted abstraction proved useful for the Sepsis Cases - Event Log since it contains less than 6k tokens.

Listing 4.1: Example of Event Log Abstraction

```
1 --- PROCESS FLOW ---
2 Leucocytes -> CRP ( frequency = 1778  performance = 20649.010 )
3              .
```

[6] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

```
            .
            .
Admission IC -> Admission IC ( frequency = 1  performance = 2424.000 )
------ PROCESS VARIANTS -------
ER Registration -> ER Triage -> ER Sepsis Triage ( frequency = 35
    performance = 1453.743 )
            .
            .
            .
ER Registration -> ER Triage -> ER Sepsis Triage -> IV Liquid ->
    Leucocytes -> CRP -> LacticAcid -> IV Antibiotics ( frequency = 2
    performance = 12127.000 )
```

In the end, the user query is enhanced with the event log abstraction; and a given instance of the dataset after going through the naive retriever has a format similar to the one presented in Listing 4.2.

Listing 4.2: Example of Dataset Instance After Going Through the Naive Retriever

```
{
        "messages": [
            {
                "role": "system",
                "content": "You are an expert process mining assistant.
                    You will receive as input:\n1. A process flow (as
                    transitions with frequency and performance).\n2. A
                    list of process variants (as full execution traces
                    with statistics).\n3. A question about the process.\n
                    \nYour task is to answer the question accurately,
                    based on the process model implied by the flow and
                    variants.\n"
            },
            {
                "role": "user",
                "content": "\n--- PROCESS FLOW ---\n\nLeucocytes -> CRP (
                     frequency = 1778  performance = 20649.010 )\n ......
                     \nAdmission IC -> Admission IC ( frequency = 1
```

```
                    performance = 2424.000 )\n\n\n\n------ PROCESS
                    VARIANTS -------\n\n ER Registration -> ER Triage ->
                    ER Sepsis Triage ( frequency = 35  performance =
                    1453.743 )\n ......\nER Registration -> ER Triage ->
                    ER Sepsis Triage -> IV Liquid -> Leucocytes -> CRP ->
                     LacticAcid -> IV Antibiotics ( frequency = 2
                    performance = 12127.000 )\n\n --- QUESTION YOU NEED
                    TO ANSWER ---\n\nInstruction: List the activities of
                    the Sepsis event log."
            }
        ]
    }
```

4.2.3 Prompting Strategies

Considering that the initial dataset and the dataset enhanced with the naive retriever are composed of different categories of non-ambiguous questions and tasks, some of them requiring a yes/no answer, others requiring a process description and everything in between (Table 4.1), we experimented with different prompting strategies to verify if there is any difference in performance during our experiments on non-ambiguous queries. Prompting strategies:

- One-Shot Prompting - in this case, for each category of questions outlined in Table 4.1, we isolated the first question-answer pair and added it to the prompt of the remaining instances of the same category (Listing 4.3). That question-answer pair was eliminated from the dataset. This led to a dataset with 79 examples.

Listing 4.3: Format of Enhanced User Query with One-Shot Prompting

```
[Process flow]
[Process variant]
Instruction: [Example instruction from the same category]
Output: [Respective example output]
Instruction: [Target instruction]
```

- Few-Shot Prompting - the use of one-shot prompting with examples from the original dataset might introduce a bias in answer generation (e.g., for the category related to listing activities, different

questions have the same answer). To address this, we also considered few-shot prompting and generated artificial question-answer pairs representative of the expected output format for each category without including any concept related to the Sepsis Cases - Event Log. For instance, for questions from category C4, we used the example below (Listing 4.4).

Listing 4.4: Format of Enhanced User Query with Few-Shot Prompting

```
1 [Process flow]
2 [Process variant]
3 Instruction: Generate a new example of a conformant trace.
4 Output: 'Order Product -> Change Product -> Pay Product-> Receive
      Product'.
5 Instruction: Create an example of a trace matching the model.
6 Output: 'Activity A -> Activity D -> Activity C'.
7 Instruction: Produce a new conforming trace.
8 Output: 'Activity B -> Activity M -> Activity W -> Activity Z ->
      Activity A'.
9 Instruction: [Target instruction from category C4]
```

It is relevant to highlight that besides the examples presented above, we also consider the same prompting strategies for user queries without retrieved context.

4.2.4 Ambiguous Dataset

Up to this point, we have presented the implementation details and datasets related to the resolution of process mining tasks when given clear queries. To assess the system's capabilities in handling ambiguous queries and to introduce the CoT component used during query disambiguation, we produced ambiguous questions. We have crafted a prompt that was provided to Grok[7] and ChatGPT[8] to automate the creation of the ambiguous questions. The produced prompt took as a starting point the questions from the initial dataset and requested the models to introduce multiple kinds of ambiguity, from pragmatic ambiguity to ambiguities associated with coreference resolution. The used prompt is presented in Listing 4.5.

Listing 4.5: Prompt for the Creation of Ambiguous Questions

```
1 [reference question]
```

[7] https://grok.com
[8] https://chatgpt.com

```
2
3 Create one or more ambiguous questions from the previous question, using
     any of the applicable methods:
4
5 - generalizing one of the concepts (more general concepts can mean
     different things);
6  - eliminating some kind of information from the question (e.g. "Is it
      possible that a trace is executed in the model?" -> "Can a trace be
      executed?");
7  - replace some nouns by pronouns (which introduces the problem of
      reference/coreference resolution);
8  - frame the question in a way that is hard to understand if the answer
      should be based on domain knowledge or event log (e.g. Non-ambiguous
      : "What happens after ER Triage?" Ambiguous: "What is the diagnosis
      after triage?");
9  - introduce ambiguity by positioning yourself as someone who is not a
      process mining analyst and therefore doesn't know the specific terms
       of process mining (e.g. "trace", "conformant", etc) and be
      realistic of questions people would actually ask.
```

Figure 4.3 presents an example of two ambiguous queries produced by Grok after receiving the prompt from Listing 4.5. It first produces a question by generalizing some terms, and then produces another question associated with coreference resolution.

18. Original: "Let's consider the Sepsis event log. Does [ER Registration] ever happen after [Leucocytes]?"

- **Ambiguous Question:** "Can registration happen after a test in the sepsis process?"
  - *Why it's ambiguous:* Generalizes "ER Registration" to "registration" and "Leucocytes" to "a test." "Sepsis process" could mean model or medical practice.
- **Ambiguous Question:** "In the sepsis records, does it come after Leucocytes?"
  - *Why it's ambiguous:* "It" replaces "ER Registration," leaving it unclear.

Figure 4.3: Example of Generated Ambiguous Queries.

After analyzing the ambiguous queries suggested by both models for each clear query from the original dataset, we removed duplicates. Note that different models might generate similar ambiguous queries from the same nonambiguous one and, at the same time, a single model might generate the same ambiguous

query for different clear queries (e.g., the ambiguous question "Tell me what happens in the sepsis data." can be mapped to the clear questions "What are the activities that happen in the sepsis cases - event log?" or "What are examples of traces that happen in the sepsis cases - event log?"). We also ensure that the ambiguous queries still mention the specific event log, considering that as a requirement for the proper behavior of the naive retriever.

This process led to an ambiguous dataset with 148 instances (contributing to the accomplishment of objective O1). The resulting dataset was not divided into categories like the initial one (Table 4.1) since some ambiguous queries can be mapped to multiple categories.

4.2.5 CoT Prompting Component

As represented in Figure 4.1, the CoT component is added to the pipeline after having the event log abstraction and it was designed to be capable of disambiguating queries like the ones generated by the procedure mentioned in Section 4.2.4. For the implementation of CoT, we crafted a prompt to guide the model through the following steps:

1. Ambiguity Identification: identify if a question is ambiguous or not.
2. Ambiguity Association: identify the terms that introduce ambiguity.
3. Non-Ambiguous Query: suggest non-ambiguous query that represents the most probable intent.
4. Final Answer.

In summary, once the pipeline receives a given user query (ambiguous or not), it retrieves the event log, produces the respective abstraction and adds the CoT steps, leading to an enhanced prompt like the one presented in Listing 4.6. This corresponds to the prompt that is received by the language model prior to answer generation.

Listing 4.6: Final Prompt Delivered to the Model (including CoT)

```
[Process flow]
[Process variant]
Instruction: [ambiguous or nonambiguous query]
To answer this question, think step by step:
Step 1: identify if the question is ambiguous.
Step 2: if the question isn't ambiguous, provide the final response. If
    the question is ambiguous, identify the terms that introduce
    ambiguity.
```

```
7 Step 3: if the question is ambiguous, suggest a non-ambiguous question
      that represents the most probable intent and provide the respective
      response.
```

4.2.6 List of All Datasets

The implementation of our system led to a constant and incremental enhancement of the prompt. With this in mind, we decided to name the datasets that resulted from the conclusion of each stage in the pipeline (Table 4.2). This nomenclature will later be used in Chapter 5 and Chapter 6 to assist in measuring the improvement in performance as additional layers of the pipeline are considered.

Table 4.2: Summary of Datasets

Dataset	Content of Prompt	No. of Instances
Initial Question-Answer Pairs	nonambiguous queries	87
Query-PM-LLM	nonambiguous queries + retrieved context	87
AmbQuery-PM-LLM	ambiguous queries + retrieved context	148
AmbQuery-PM-LLM + CoT	ambiguous queries + retrieved context + CoT steps	148

The produced prompting strategies (one-shot prompting and few-shot prompting for questions with and without retrieved context) were used in the experiments but were not added to the pipeline.

5

Experimental Setup

Contents

This chapter outlines the metrics used to evaluate the developed system. These include metrics for assessing correctness, as well as metrics for evaluating how reasoning steps reduce uncertainty and resolve ambiguity in responses to queries. Additionally, we present the experimental setup for the results discussed in Chapter 6.

5.1 Evaluation Metrics

In NLP, particularly for tasks like machine translation, text summarization and question answering, automated evaluation metrics are essential for assessing the quality of generated text against reference texts. In the context of the thesis, they will be used to evaluate the quality of the generated answer, given the reference answer present in the formulated datasets. As our initial metrics, we will be using BLEU, ROUGE, and BERTScore, which are described in Section 5.1.1, Section 5.1.2, and Section 5.1.3, respectively.

Additionally, since our system was designed to deal with ambiguity, it should be able to reduce the uncertainty regarding the generated answer when applying the reasoning steps. To measure such reduction, we will employ a form of semantic entropy over the generated response. In Section 5.1.4, we define semantic entropy and present the results from replicating the original experiment by Kuhn et al. [45] for its computation. We also outline the adjustments and assumptions we made for our datasets.

5.1.1 BLEU

BLEU, which stands for Bilingual Evaluation Understudy, is a widely used metric for evaluating machine translation systems. Introduced by Papineni et al. [46], BLEU measures the similarity between machine-generated translations and one or more human reference translations by computing the n-gram precision.

The BLEU score is calculated as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{5.1}$$

where:

- p_n is the modified n-gram precision,
- BP is the brevity penalty to penalize short translations,
- w_n are uniform weights (typically $w_n = 1/N$),
- N is the maximum n-gram order (often 4).

While BLEU was originally proposed for machine translation, it has since been adopted as a general-purpose metric for natural language generation tasks such as text summarization and question answering. Although it does not capture deeper semantic equivalence or handle paraphrasing effectively, it provides a useful baseline for measuring surface-level correspondence between generated and expected responses. In particular, "Evaluating Question Answering Evaluation" [47] discusses BLEU as one of the standard metrics for assessing QA models.

5.1.2 ROUGE

ROUGE, which stands for Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics primarily used for evaluating automatic summarization and machine translation. Developed by Lin [48], ROUGE includes the following variants:

- ROUGE-N: N-gram recall metric that evaluates the overlap between a candidate summary and

reference summaries. The ROUGE-N score for n-grams is:

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{References}} \sum_{\text{gram}_n \in S} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{S \in \text{References}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)}, \tag{5.2}$$

where $\text{Count}_{\text{match}}(\text{gram}_n)$ is the number of n-grams that match between the candidate and reference summaries.

- ROUGE-L: Longest Common Subsequence (LCS)-based F-measure, which captures sentence-level structure.

ROUGE is effective for summarization but may undervalue semantic equivalence. Considering QA settings and in similarity to BLEU, ROUGE is discussed in the work of Chen et al. [47].

5.1.3 BERTScore

BERTScore is a metric that leverages contextual embeddings from PLMs like BERT [22] to evaluate text generation quality. Proposed by Zhang et al. [49], it computes similarity based on token-level embeddings rather than exact matches.

BERTScore consists of precision, recall, and F1 scores, computed as:

$$P = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \langle \hat{x}_j, x_i \rangle, \tag{5.3}$$

$$R = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \langle x_i, \hat{x}_j \rangle, \tag{5.4}$$

where x and $\hat{x}$ are the token embeddings for reference and candidate sentences, respectively, and $\langle \cdot, \cdot \rangle$ denotes cosine similarity.

The work of Chen et al. [47], which studied metrics like BLEU and ROUGE, also highlighted BERTScore's ability to capture paraphrases and semantic similarity. This underscores BERTScore's suitability for diverse NLP tasks beyond surface-level overlaps.

5.1.4 Semantic Entropy

To quantitatively assess the level of ambiguity in user queries and the effectiveness of using CoT for disambiguation, we employ semantic entropy as a key metric. Semantic entropy, introduced by Kuhn et al. [45] in 2023, provides a measure of uncertainty in natural language generation that accounts for linguistic invariances, focusing on the diversity of meanings. Semantic entropy is computed as follows:

1. For a given query, multiple answers are sampled from a given LLM model. Those samples are generated with a greedy approach (i.e., generating with `do_sample = True` and `num_beams = 1`).

2. These samples are then clustered into semantic equivalence classes using a bidirectional entailment classifier (DeBERTa large model [50] fine-tuned with MNLI task[1]), which groups outputs that mutually entail each other (i.e., one can be inferred from the other, capturing shared meanings).

3. The entropy is then estimated over the probability distribution of these clusters. Formally, if C represents the set of semantic clusters and Equation 5.5 is the semantic likelihood for cluster c given input x, then semantic entropy is given by Equation 5.6.

$$p(c \mid x) = \sum_{s \in c} p(s \mid x) \tag{5.5}$$

$$H_{\text{sem}}(x) = -\sum_{c \in \mathcal{C}} p(c \mid x) \log p(c \mid x) \tag{5.6}$$

In practice, Equation (5.6) measures the unpredictability of the meanings of generated answers: low values (near 0) indicate concentrated probability on one cluster, signaling unambiguous outputs; high values reflect even spread across clusters, highlighting diverse or conflicting interpretations and thus ambiguity.

To implement the computation of this metric, we started by replicating the original experiment presented by Kuhn et al. [45]. This was followed by adapting the setup to consider our datasets. Furthermore, we experimented using DeBERTa and Llama for bidirectional entailment to see which option provided more accurate values of semantic entropy in our case. This is described in detail in Section 5.1.4.A.

5.1.4.A Implementation of Semantic Entropy Computation

To quantify the ambiguity and uncertainty inherent in our AmbQuery-PM-LLM dataset, as well as its reduction when considering Amb-Query-PM-LLM + CoT, we resorted to the computation of semantic entropy, following the formulation by Kuhn et al. [45].

The code with the implementation of their work is available on GitHub[2] (deprecated). The experiments they ran used OPT [51] for answer generation, and considered two different datasets: CoQA [52] and TriviaQA [53]. Taking that as a starting point, we subsampled 10% of CoQA for replication of the original experiment using Google Colab[3] with an A100 GPU to accelerate LLM inference. We substituted OPT with Meta-Llama-3-8B-Instruct (our thesis's primary model) and retained key parameters as presented in Table 5.1.

Given the deprecation, we validated the first two pipeline steps (sample generation and clustering via DeBERTa) and made minor adjustments in the calculation of Semantic Set IDs (which are the variables

[1] https://huggingface.co/microsoft/deberta-large-mnli
[2] https://github.com/lorenzkuhn/semantic_uncertainty
[3] https://colab.google

Table 5.1: Setup for Replication of Original Experiment on Semantic Entropy Computation

Parameter	Value
Model	Llama-3-8B-Instruct
Samples per Question	10
do_sample	`True`
num_beams for Sample Generation	1
temperature	0.5
ROUGE-L Threshold for Answer Correctness	> 0.3

that define and differentiate the clusters/semantically equivalent classes).

After the minor changes, we ran the whole pipeline and reached the following values:

- Predictive Entropy AUROC for CoQA: 0.59971
- Semantic Entropy AUROC for CoQA: 0.61093

Within the computation of semantic entropy, the AUROC metric specifically assesses the quality of uncertainty estimation and evaluates how well an uncertainty score (e.g., from semantic entropy) can predict whether a language model's generated answer is correct or incorrect, framing the problem as a binary classification: "trust the answer" (low uncertainty for correct outputs) versus "do not trust" (high uncertainty for incorrect ones). Considering the AUROC values presented above, semantic entropy was identified as a better metric than the baseline (e.g., predictive entropy), which goes in line with the results obtained by [45]. This motivated us to run the experiment on our datasets.

One of the main challenges in relation to our datasets corresponds to long-format answers that usually mention activities even when their primary goal is to describe a process or provide a trace, which contributes to making the clustering less obvious. To increase our confidence in the use of semantic entropy as metric, we decided to also experiment using Llama for bidirectional entailment and compare it with DeBERTa. The use of Llama for bidirectional entailment corresponds to considering all possible pairs within the 10 generated samples for a given question, prompting the model to look at the pair `<sample_x, sample_y>` and asking if `sample_x` entails or contradicts `sample_y` and vice versa. Samples that entail each other are clustered together.

After running both experiments on our ambiguous dataset (Amb-Query-PM-LLM), we obtained the following results:

- Semantic Entropy AUROC using DeBERTa for bidirectional entailment: 0.73287
- Semantic Entropy AUROC using Llama for bidirectional entailment: 0.69580

Considering the obtained AUROC values, DeBERTa was a better classifier of bidirectional entailment on our Amb-Query-PM-LLM dataset and that was the reason we opted to maintain it for the computation of our final semantic entropy results presented in Section 6.3. The values outlined in Table 5.1 were also used during our experiments.

5.2 Experimental Setup

For all of our experiments, the model used for answer generation corresponds to Meta-Llama-3-8B-Instruct. The initial experiments, which still don't consider ambiguity, were performed on some of the datasets outlined in Table 4.2:

- Initial Question-Answer Pairs considering the prompting strategies (which was enough to show the poor performance when disregarding the context).
- Query-PM-LLM with and without prompting strategies to see how each influenced the output.

For those initial experiments, we considered BLEU, ROUGE and BERTScore as presented in Table 5.2. Furthermore, to select the optimal `top_p` and `temperature` values for each of the datasets of the initial experiments, we started by conducting a hyperparameter search (presented in Appendix A). The hyperparameter search led to the following values for each case:

- Initial Question-Answer Pairs + One-Shot Prompting: `<temperature = 0.9,top_p = 0.7>`
- Initial Question-Answer Pairs + Few-Shot Prompting: `<temperature = 0.7,top_p = 0.9>`
- Query-PM-LLM: `<temperature = 0.7,top_p = 0.3>`
- Query-PM-LLM + One-Shot Prompting: `<temperature = 0.7,top_p = 0.7>`
- Query-PM-LLM + Few-Shot Prompting: `<temperature = 0.7,top_p = 0.7>`

After the initial experiments, we computed and compared the semantic entropy of Query-PM-LLM, AmbQuery-PM-LLM, and AmbQuery-PM-LLM + CoT (datasets also outlined in Table 4.2) to see how uncertainty changed in each scenario.

Table 5.2: Summary of Metrics used on Each Dataset

Dataset	Metrics Used for Evaluation
Initial Question-Answer Pairs	BLEU, ROUGE, BERTScore
Query-PM-LLM	BLEU, ROUGE, BERTScore, Semantic Entropy
AmbQuery-PM-LLM	Semantic Entropy
AmbQuery-PM-LLM + CoT	Semantic Entropy

6

Experimental Results

Contents

In this chapter, we present results following the experimental setup previously outlined and considering the metrics described in Chapter 5. We start by presenting correctness results on the Initial Question-Answers Pairs, followed by the set of results when adding the naive retriever (Query-PM-LLM). We finalize by collectively showing the entropy results when considering Query-PM-LLM, AmbQuery-PM-LLM and AmbQuery-PM-LLM + CoT.

6.1 Results on Initial Question-Answer Pairs + Prompting Strategies

Without the event log abstraction, considering both prompting strategies, the model struggled to produce answers and tended to ask for more information regarding the sepsis log. This is validated by the results presented in Table 6.1 and shows that the selected model has limited knowledge of the content of the sepsis cases - event log, which is a good first indication of the need to add context. It is relevant to highlight

that while the model has no knowledge of the log itself, it occasionally showed general knowledge of concepts associated with sepsis.

Table 6.1: Summary of Results on Initial Question-Answer Pairs

Metric	One-Shot Prompt (No Context)	Few-Shot Prompt (No Context)
Average ROUGE-1	0.0815	0.0402
Average ROUGE-2	0.0401	0.0044
Average ROUGE-L	0.0672	0.0302
Average BLEU	0.0336	0.0017
Average BERTScore	0.3721	0.3425

6.2 Results on Query-PM-LLM With and Without Prompting Strategies

After obtaining results on the Initial Question-Answer Pairs, we proceeded to see how the addition of the event log abstraction changed the performance. For Query-PM-LLM, the model can correctly list the set of activities (category 1), identify some of the next possible activities (category 7), and properly list traces (categories 3 and 4), which is consistent with the results in Figure 6.1. For process description (category 2), the model provides varied responses (ROUGE-1: 0.24), sometimes describing the process correctly, other times listing process variants when it should provide only the process description.

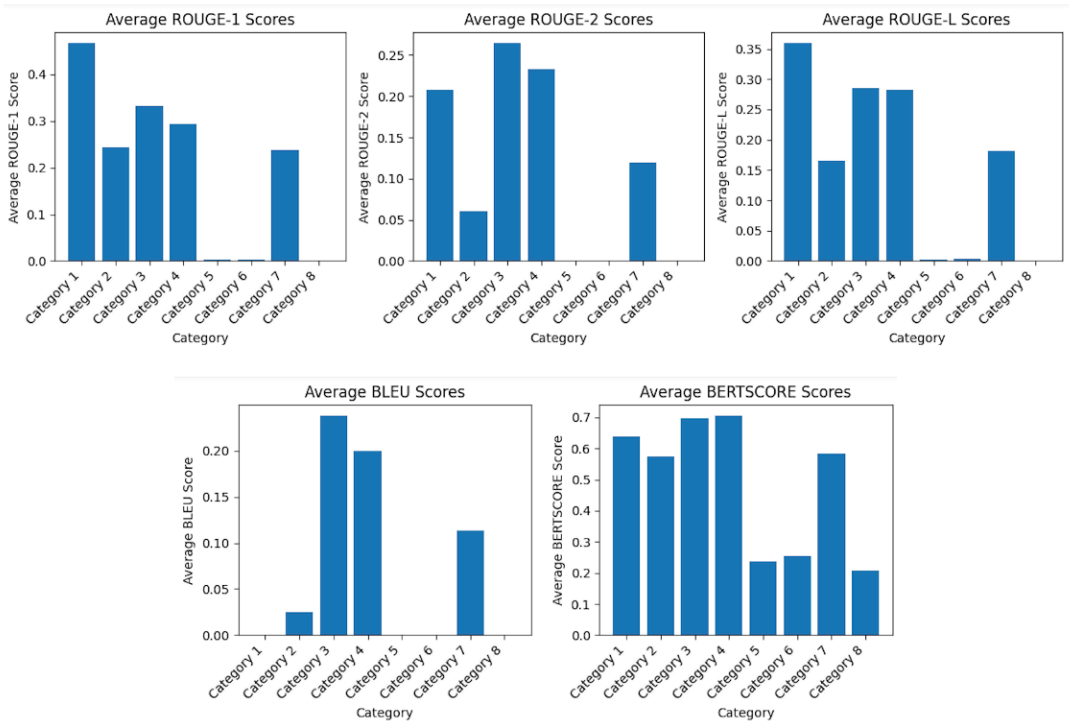

Figure 6.1: Results: Query-PM-LLM.

Categories 5, 6, and 8 correspond to yes/no questions, where the model's tendency to predict the next token during generation leads to long-format analytical responses instead of concise "Yes" or "No" answers. This makes text generation metrics (ROUGE, BLEU, BERTScore) inappropriate for evaluating these categories. At the same time, even with `max_new_tokens=275`, the model typically begins with lengthy analysis ("Based on the provided process flow and variants, I can analyze...") before attempting to answer, rarely mentioning "Yes" or "No" within the generated output. This leads to 62.7% of yes/no questions (32/51) producing unclear responses. Such poor performance in those categories was one of the motivations behind the experiments with different prompting strategies added to Query-PM-LLM - aimed at providing examples that represented the expected output format.

For Query-PM-LLM + One-Shot Prompt and Query-PM-LLM + Few-Shot Prompt, Llama properly answers categories 1, 3, 4 and 7 in similarity to Query-PM-LLM without prompting strategy. For process description (category 2), the model is able to properly identify the intent of the question and provide the appropriate answer, contributing to better results in regards to Query-PM-LLM without prompting strategies.

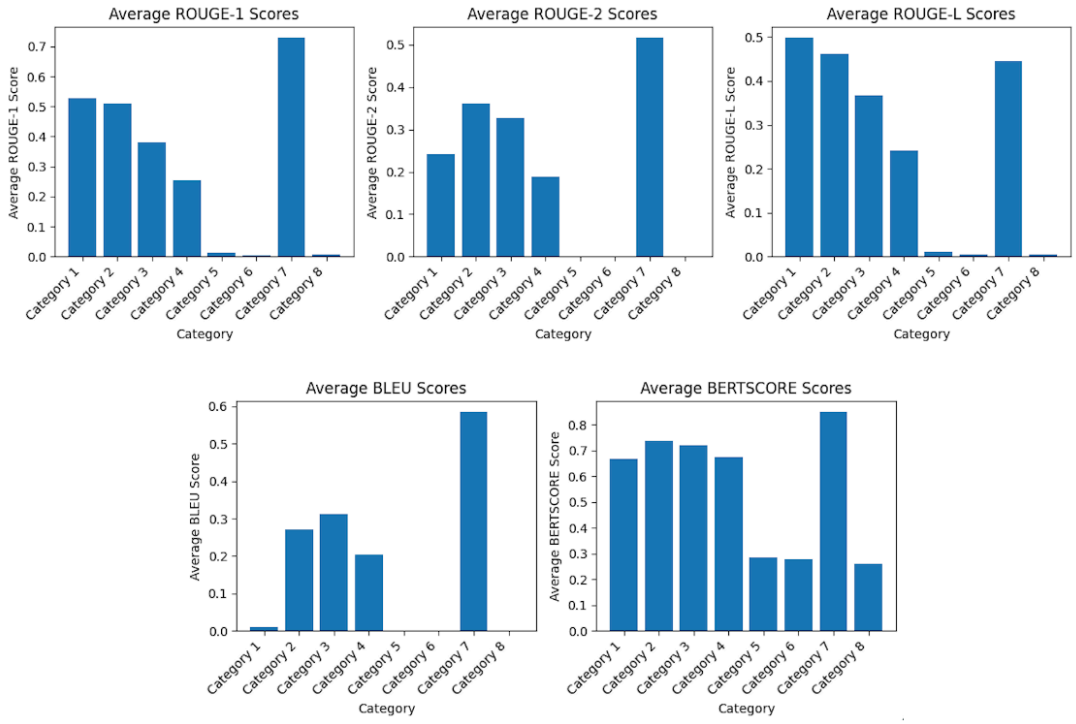

Figure 6.2: Results: Query-PM-LLM + One-Shot Prompt.

As it is verified in Figure 6.2 and Figure 6.3, the results are slightly better for one-shot prompting than few-shot prompting since the exemplar in one-shot is from the dataset and the exemplars on the few-shot prompting are more generic.

Considering the categories 5, 6, and 8, which require a "Yes" or "No" answer:

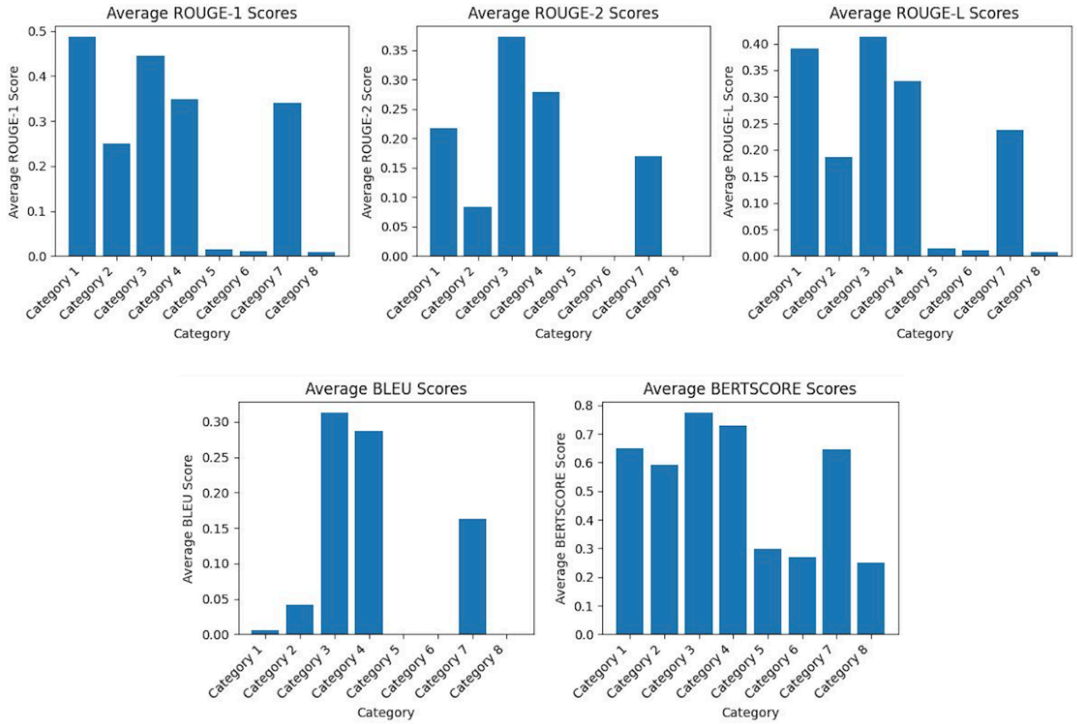

Figure 6.3: Results: Query-PM-LLM + Few-Shot Prompt.

- The model still produced unclear responses in 22,9% of the cases in Query-PM-LLM + One-Shot Prompt. Considering the questions for which the model presented clear answers, the accuracy was 50.0%.

- The model still produced unclear responses in 19,6% of the cases in Query-PM-LLM + Few-Shot Prompt. Considering the questions for which the model presented clear answers, the accuracy was 52.9%.

In summary, and as presented in table Table 6.2, both one-shot prompting and few-shot prompting led to better performance when compared with Query-PM-LLM without prompting strategy with Query-PM-LLM + One-Shot Prompt showing the best results.

Table 6.2: Summary of Results on Query-PM-LLM Considering Different Prompting Strategies

Metric	No Prompting Strategy	One-Shot Prompt	Few-Shot Prompt
Average ROUGE-1	0.1299	0.1892	0.1542
Average ROUGE-2	0.0652	0.1252	0.0828
Average ROUGE-L	0.1022	0.1702	0.1289
Average BLEU	0.0385	0.0943	0.0543
Average BERTScore	0.4006	0.4519	0.4359

6.3 Results on Entropy Experiments

As presented in Table 6.3, entropy results show that the ambiguous dataset without CoT (AmbQuery-PM-LLM) has the highest entropy, followed by Query-PM-LLM. The semantic entropy on the ambiguous dataset with CoT (AmbQuery-PM-LLM + CoT) is the lowest of all.

Table 6.3: Entropy Results

Dataset	Average Semantic Entropy
Query-PM-LLM	0.996114
AmbQuery-PM-LLM	1.128019
AmbQuery-PM-LLM + CoT	0.413015

It was expected that the semantic entropy of AmbQuery-PM-LLM + CoT would reduce when compared to the semantic entropy of AmbQuery-PM-LLM, as the reasoning steps would guide the model to the answer and reduce diversity in the output. However, when looking at Query-PM-LLM's entropy, it was unexpected that the value would be significantly higher than the value of AmbQuery-PM-LLM + CoT's entropy. Examining specific examples from each of the datasets and analyzing the generated samples, it was noticeable that the main contribution of CoT didn't specifically come in terms of query disambiguation, but rather in terms of general consistency. CoT increased robustness since the generated samples, which were generated with a greedy approach (single beam), were consistent among them. On the other hand, while the initially generated answers on Query-PM-LLM are reasonably good, there is higher diversity when following a greedy approach in the generation of multiple samples, suggesting that the addition of 'think step by step' to the prompt proves useful in increasing the model's confidence in the delivery of an answer, regardless of the existence of ambiguity. To better understand this, let us consider the examples presented below.

For the nonambiguous question 'list the activities in the sepsis event log', while the most likely generation was the list of activities, when following the greedy approach to generate samples and disregarding CoT, the model sometimes provided traces. This led to a semantic entropy of 0.667.

Consider the ambiguous and general question 'I'd like to know about the sepsis process.' Without CoT, there was diversity among the answers, some of them relying on the event log itself, and others relying on the limited general knowledge of sepsis instead (which is not ideal since the goal is the resolution of process mining tasks and therefore the event log should be the basis for the answer). This led to a semantic entropy of 0.8241. For that same example, when using CoT, the semantic entropy reduced to 0.1616, the generated samples were always based on the event log data and usually presented lists of activities in an order in which they could occur.

7

Conclusion

Contents

In the context of this dissertation, we implemented a system that starts by receiving a user query, retrieves an event log from an online repository and obtains its respective abstraction. It finalizes by adding CoT steps to guide the model in disambiguating the user query. This system replicates the behavior of a conversational agent for process mining that is aimed at dealing with ambiguity as outlined in contribution EC1. We also introduced various datasets that were used to test the system: Initial Question-Answer Pairs which, with our naive retriever, led to Query-PM-LLM; AmbQuery-PM-LLM which was built by introducing specific types of ambiguity in the original questions. AmbQuery-PM-LLM when used with the CoT component, led to AmbQuery-PM-LLM + CoT. Those datasets (mainly Query-PM-LLM and AmbQuery-PM-LLM) fulfil contribution EC3. Through our evaluation process, we have seen how the addition of a process abstraction contributed for answer generation and highlighted that the publicly available sepsis cases - event log, which is a large one, is not part of the parametric memory of Meta-LLama-3-8B-Instruct. At the same time, we have seen how different prompting strategies influenced the model's behavior in answer generation.

We have finalized by computing the entropy values on Query-PM-LLM, AmbQuery-PM-LLM and

AmbQuery-PM-LLM + CoT. From those results, we have seen the CoT's usefulness in reducing uncertainty in answer generation even when there is no ambiguity in a given user query. The interesting results of the CoT experiments contributed to the fulfilment of contribution EC2.

7.1 Limitations and Future Work

The main limitation of our system corresponds to the need to mention a specific event log when formulating a question. This was initially done to simplify the retrieval component while still allowing query disambiguation in the CoT component. Future iterations could include the implementation of CoT before the retrieval and abstraction of the event log. This way, the reasoning steps would disambiguate the query and identify the event log that is relevant for retrieval, contributing to a more refined RAG-LLM. It would still differentiate itself from RQ-RAG [25], since RQ-RAG [25] does not include reasoning steps. Even after retrieval, CoT could still play an important role.

A related constraint stems from the abstraction granularity: our fixed abstraction (DFG relations and process variants via pm4py) effectively condenses the Sepsis Cases event log within the 8k-token context limit of Meta-Llama-3-8B-Instruct but risks omitting critical details, such as timestamps or resources. This static approach assumes uniform needs across queries and logs, limiting the system to only be capable of answering process mining questions belonging to the categories outlined in Table 4.1. To mitigate this, future work could include the use of reasoning steps for query disambiguation and after identifying the user's intent, delegate the task to a reasoner.

The other main limitation is not about the system, but about the evaluation process. While we evaluate and see reduction of uncertainty in AmbQuery-PM-LLM + CoT when compared to AmbQuery-PM-LLM, we still need to define a formal method to evaluate the correctness (i.e., while the reasoning steps converged and reduced entropy, we did not formally evaluate if they converged to the right answer). In the future, this evaluation could be done with the assistance of a human evaluator.

Bibliography

[1] M. Dumas, M. L. R. J. Mendling, and H. A. Reijers, Fundamentals of Business Process Management, 2nd ed. Springer Berlin, Heidelberg, 2018.

[2] Z. Wang, Z. Chu, T. V. Doan, S. Ni, M. Yang, and W. Zhang, "History, development, and principles of large language models-an introductory survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.06853

[3] A. Martins, F. Melo, and M. Figueiredo, "Deep Learning Course, Instituto Superior Tecnico," Unpublished, 2023-2024, lecture slides.

[4] Y. Fontenla-Seco, S. Winkler, A. Gianola, M. Montali, M. Lama, and A. Bugarín-Diz, "The droid you're looking for: C-4pm, a conversational agent for declarative process mining," The BPM 2023 Demos and Resources Forum. 21st International Conference on Business Process Management, pp. 112–116, 2023. [Online]. Available: https://ceur-ws.org/Vol-3469/paper-20.pdf

[5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 34th Conference on Neural Information Processing Systems, 2020. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[6] M. Weske, Business Process Management - Concepts, Languages, Architectures, 2nd Edition. Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-28616-2

[7] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-662-49851-4

[8] J. Becker, R. Fischer, and C. Janiesch, "Optimizing u.s. health care processes - a case study in business process management," AMCIS 2007 PROCEEDINGS, vol. 504, December 2007. [Online]. Available: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=2014&context=amcis2007

[9] L. Pufahl, F. Zerbato, B. Weber, and I. Weber, "BPMN in healthcare: Challenges and best practice," Information Systems, vol. 107, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437922000217

[10] Y. F. Seco, "Process-to-text: A framework for the automatic generation of natural language descriptions of processes," Ph.D. dissertation, Escola de Douturamento Internacional da USC, 2023.

[11] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang, "On the opportunities and risks of foundation models," 2022. [Online]. Available: https://arxiv.org/abs/2108.07258

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[13] U. Jessen, M. Sroka, and D. Fahland, "Chit-chat or deep talk: Prompt engineering for process mining," arXiv e-prints, jul 2023. [Online]. Available: https://drive.google.com/file/d/1R3btI3q3CwfAfJ6-eXbHlEuGllF0srn6/view

[14] A. Keluskar, A. Bhattacharjee, and H. Liu, "Do llms understand ambiguity in text? a case study in open-world question answering," November 2024. [Online]. Available: https://arxiv.org/html/2411.12395v1

[15] C. Di Ciccio and M. Montali, "Declarative process specifications: Reasoning, discovery, monitoring," in Process Mining Handbook, ser. Lecture Notes in Business Information Processing, W. M. P. van der Aalst and J. Carmona, Eds. Springer, Cham, 2022, vol. 448, pp. 108–152.

[16] G. De Giacomo and M. Y. Vardi, “Linear temporal logic and linear dynamic logic on finite traces,” p. 854–860, 2013.

[17] W. M. P. van der Aalst, “Process mining: A 360 degree overview,” in Process Mining Handbook, ser. Lecture Notes in Business Information Processing, W. M. P. van der Aalst and J. Carmona, Eds. Springer, Cham, 2022, vol. 448, pp. 1–34.

[18] D. Jurafsky and J. H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd ed., 2024. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[19] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[20] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” CoRR, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[21] S. I. Gallant, “Context vectors: A step toward a ”grand unified representation”,” in Hybrid Neural Systems, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:43641558

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long and Short Papers), pp. 4171–4186, June 2019. [Online]. Available: https://aclanthology.org/N19-1423/

[23] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” CoRR, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[24] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[25] C.-M. Chan, C. Xu, R. Yuan, H. Luo, W. Xue, Y. Guo, and J. Fu, "RQ-RAG: Learning to Refine Queries for Retrieval Augmented Generation," 2024. [Online]. Available: https://arxiv.org/abs/2404.00610

[26] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," Advances in Neural Information Processing Systems, vol. 35, 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[27] S. E. of Philosophy. (2021) Ambiguity. First published May 16, 2011; substantive revision May 22, 2021. [Online]. Available: https://plato.stanford.edu/eNtRIeS/ambiguity/

[28] J. Ginzburg, "Interrogatives: Questions, facts and dialogue," in The Handbook of Contemporary Semantic Theory, S. Lappin, Ed. Oxford: Blackwell, 1996, pp. 359–423.

[29] M. Franceschetti, R. Seiger, H. A. López, A. Burattin, L. García-Bañuelos, and B. Weber, "A characterisation of ambiguity in bpm," in Conceptual Modeling: 42nd International Conference, ER 2023, Lisbon, Portugal, November 6–9, 2023, Proceedings. Berlin, Heidelberg: Springer-Verlag, 2023, p. 277–295. [Online]. Available: https://doi.org/10.1007/978-3-031-47262-6_15

[30] R. J. C. Bose, R. S. Mans, and W. M. van der Aalst, "Wanna improve process mining results?" in 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2013, pp. 127–134.

[31] H. Kourani, A. Berti, D. Schuster, and W. M. P. van der Aalst, "ProMoAI: Process modeling with generative ai," Proceedings of the Thirty-ThirdInternational Joint Conference on Artificial Intelligence, August 2024. [Online]. Available: http://dx.doi.org/10.24963/ijcai.2024/1014

[32] J. Kopke and A. Safan, "Efficient LLM-Based conversational process modeling," 2024. [Online]. Available: https://drive.google.com/file/d/1EVl0SVKeyTnsw6pb59WgvL1K8Y88weRk/view

[33] A. Rebmann, F. D. Schmidt, G. Glavaš, and H. van der Aa, "Evaluating the ability of LLMs to solve semantics-aware process mining tasks," 2024. [Online]. Available: https://arxiv.org/abs/2407.02310

[34] H. Kourani, A. Berti, J. Hennrich, W. Kratsch, R. Wei-dlich, C.-Y. Li, A. Arslan, W. M. P. van der Aalst, and D. Schuster, "Leveraging large language models for enhanced process model com- prehension," 2024. [Online]. Available: https://arxiv.org/pdf/2408.08892

[35] A. Berti, D. Schuster, and W. M. P. van der Aalst, "Abstractions, scenarios, and prompt definitions for process mining with llms: A case study," 2023. [Online]. Available: https://arxiv.org/abs/2307.02194

[36] A. Ferrari, G. Lipari, S. Gnesi, and G. O. Spagnolo, “Pragmatic ambiguity detection in natural language requirements,” 2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), pp. 1–8, 2014.

[37] A. Ferrari and A. Esuli, “A NLP approach for cross-domain ambiguity detection in requirements engineering,” Automated Software Engineering, vol. 26, p. 559–598, June 2019. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s10515-019-00261-7.pdf

[38] M. Guo, M. Zhang, S. Reddy, and M. Alikhani, “Abg-coqa: Clarifying Ambiguity in Conversational Question Answering,” Automated Knowledge Base Construction, 2021.

[39] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, “BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” CoRR, vol. abs/1910.13461, 2019. [Online]. Available: http://arxiv.org/abs/1910.13461

[40] V. Karpukhin, B. Oguz, S. Min, L. Wu, S. Edunov, D. Chen, and W. Yih, “Dense passage retrieval for open-domain question answering,” CoRR, vol. abs/2004.04906, 2020. [Online]. Available: https://arxiv.org/abs/2004.04906

[41] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open Foundation and Fine-Tuned Chat Models,” 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[42] G. Feng, B. Zhang, Y. Gu, H. Ye, D. He, and L. Wang, “Towards revealing the mystery behind chain of thought: A theoretical perspective,” Advances in Neural Information Processing Systems, vol. 36, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/dfc310e81992d2e4cedc09ac47eff13e-Abstract-Conference.html

[43] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, “Training verifiers to solve math word problems,” CoRR, vol. abs/2110.14168, 2021. [Online]. Available: https://arxiv.org/abs/2110.14168

[44] A. Talmor, J. Herzig, N. Lourie, and J. Berant, "CommonsenseQA: A question answering challenge targeting commonsense knowledge," Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4149–4158, June 2019. [Online]. Available: https://aclanthology.org/N19-1421/

[45] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," 2023. [Online]. Available: https://arxiv.org/abs/2302.09664

[46] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, P. Isabelle, E. Charniak, and D. Lin, Eds. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. [Online]. Available: https://aclanthology.org/P02-1040/

[47] A. Chen, G. Stanovsky, S. Singh, and M. Gardner, "Evaluating question answering evaluation," in Proceedings of the 2nd Workshop on Machine Reading for Question Answering, A. Fisch, A. Talmor, R. Jia, M. Seo, E. Choi, and D. Chen, Eds. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 119–124. [Online]. Available: https://aclanthology.org/D19-5817/

[48] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in Text Summarization Branches Out. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013/

[49] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," 2020. [Online]. Available: https://arxiv.org/abs/1904.09675

[50] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," 2021. [Online]. Available: https://arxiv.org/abs/2006.03654

[51] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022. [Online]. Available: https://arxiv.org/abs/2205.01068

[52] S. Reddy, D. Chen, and C. D. Manning, "CoQA: A conversational question answering challenge," Transactions of the Association for Computational Linguistics, vol. 7, pp. 249–266, 2019. [Online]. Available: https://aclanthology.org/Q19-1016/

[53] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," 2017. [Online]. Available: https://arxiv.org/abs/1705.03551

A

Hyperparameter Search

For the selection of `top_p` and `temperature` values for answer generation considering Initial Question-Answer Pairs with prompting strategies and Query-PM-LLM with and without prompting strategies, we conducted a hyperparameter search. For each of the datasets, four examples were selected - one from each of the first four categories. We saw the performance of the model on a given combination of `<query,temperature,top_p>` and then averaged each metric over the four selected queries considering the same `<temperature,top_p>` combination. This way, for each dataset, we obtained 5 heatmaps, each corresponding to the performance according to one of 5 metrics, as presented in the example in Figure A.1.

With this setup and considering all the obtained results, the selected `<temperature,top_p>` values correspond to those that tended to yield better results in most metrics. Taking into account the heatmaps presented for Query-PM-LLM in Figure A.1, the selected combination of hyperparameter values for that dataset correspond to `<temperature = 0.7,top_p = 0.3>`. The same analysis was done for the remaining cases.

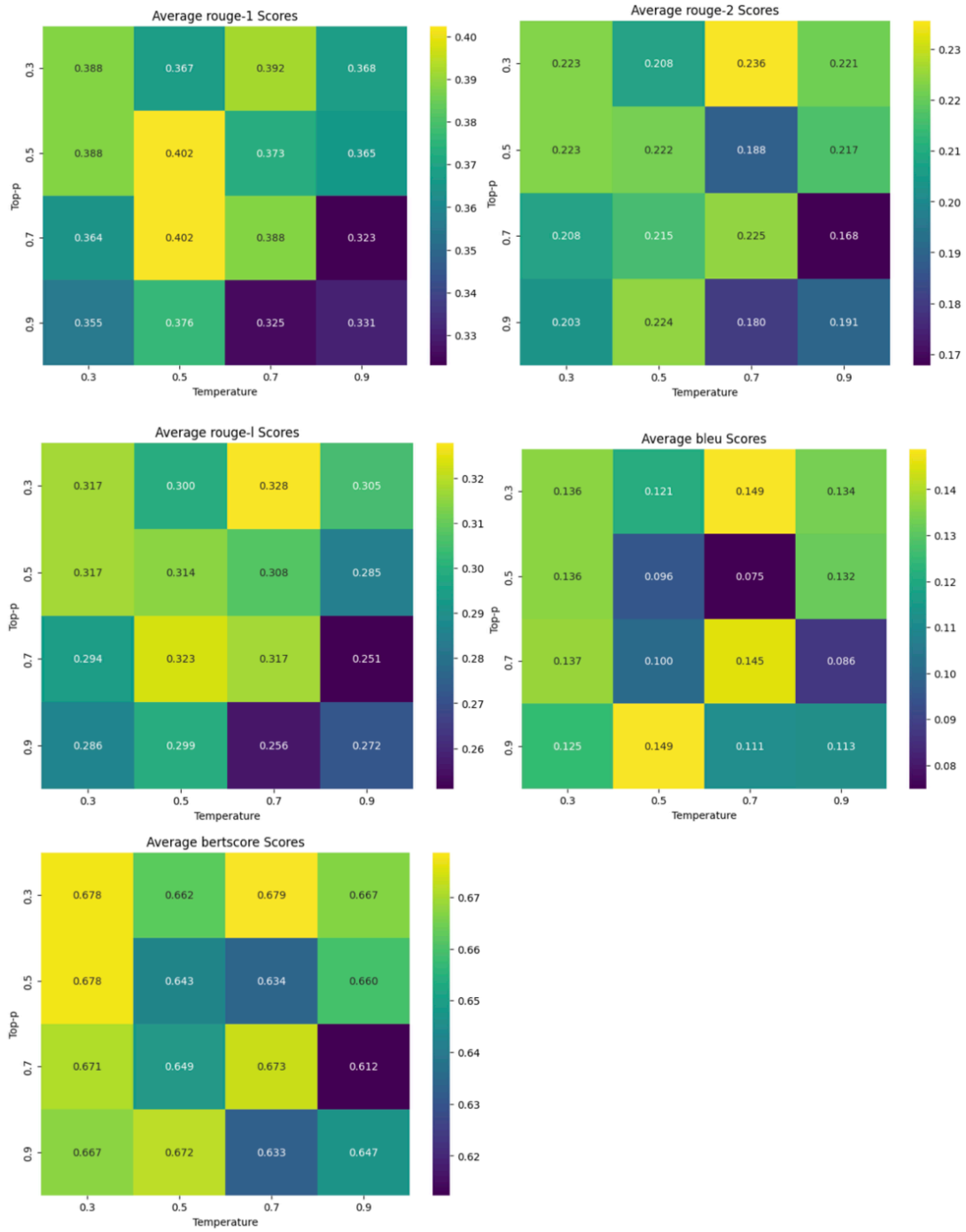

Figure A.1: Heatmaps for `<temperature,top_p>` combinations on Query-PM-LLM.