universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Enhancing Process Mining with LLMs by Applying Instruction Tuning

verfasst von | submitted by

Vira Pyrih

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von | Supervisor:	Univ.-Prof. Han van der Aa Ph.D.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Han van der Aa, for the continuous guidance, motivation, and encouragement throughout this research. Your insightful feedback and thoughtful direction made this thesis possible.

I am also thankful to Adrian Rebmann, whose expertise, responsiveness, and valuable input greatly contributed to the development and refinement of this work.

To my family and friends—thank you for your unwavering support, patience, and belief in me throughout this journey. A special thank you to my boyfriend, Roham, for always being there for me with kindness and emotional support.

Lastly, I wish to express my heartfelt gratitude to the defenders of Ukraine. Your courage and sacrifice have allowed me to focus on my studies, knowing that my family and homeland are being protected.

Abstract

The interest in process mining has grown as organizations increasingly take advantage of event data to gain insight into their processes. Traditionally, process mining relies on statistical methods to analyze event logs. However, recent advancements in Large Language Models (LLMs) offer a novel alternative, emphasizing semantic understanding without depending heavily on historical context. Instruction tuning, the practice of training models to follow natural language instructions, shows significant potential for process mining. It enables models to generalize within a domain, reducing the need for repeated fine-tuning across tasks while improving their ability to comprehend user prompts.

This thesis explores the potential of instruction tuning in process mining supported by the construction of a domain-specific instruction dataset. The dataset is organized into three task clusters: anomaly detection, prediction, and discovery. These clusters are built from five process mining tasks identified in prior research. We train open-source LLMs from the Llama and Mistral families using a leave-one-cluster-out strategy, where each cluster is excluded in turn to evaluate generalization.

We evaluate model performance using macro F_1 scores for classification tasks and footprint-based fitness for generation tasks. Results show that instruction tuning consistently improves generalization in the prediction and discovery clusters, particularly with models exceeding 60B parameters, while effects on anomaly detection remain less conclusive. In addition to quantitative metrics, our qualitative analysis shows that instruction-tuned models follow formatting constraints more reliably and produce more interpretable responses compared to base models.

Our findings demonstrate that instruction tuning is a promising approach for developing flexible, semantics-aware models in the process mining domain.

Kurzfassung

Das Interesse am Process Mining ist gewachsen, da Organisationen zunehmend Ereignisdaten nutzen, um Einblicke in ihre Prozesse zu gewinnen. Traditionell stützt sich Process Mining auf statistische Methoden zur Analyse von Ereignisprotokollen. Jüngste Fortschritte im Bereich der Large Language Models (LLMs) bieten jedoch eine neuartige Alternative, bei der das semantische Verständnis im Vordergrund steht, ohne stark auf historischen Kontext angewiesen zu sein. Instruction Tuning,das Trainieren von Modellen zur Befolgung natürlicher Sprachinstruktionen, zeigt großes Potenzial für das Process Mining. Es ermöglicht Modellen, innerhalb einer Domäne zu generalisieren, wodurch der Bedarf an wiederholtem Fine-Tuning für einzelne Aufgaben reduziert wird und gleichzeitig das Verständnis für nutzerseitige Anweisungen verbessert wird.

Diese Arbeit untersucht das Potenzial von Instruction Tuning im Process Mining, gestützt durch die Erstellung eines domänenspezifischen Instruction-Datensatzes. Der Datensatz ist in drei Aufgabencluster gegliedert: Anomalieerkennung, Vorhersage und Discovery. Diese Cluster basieren auf fünf Process-Mining-Aufgaben, die in früheren Arbeiten identifiziert wurden. Wir trainieren quelloffene LLMs der Llama- und Mistral-Familien mithilfe einer Leave-One-Cluster-Out-Strategie, bei der jeweils ein Cluster ausgeschlossen wird, um die Generalisierungsfähigkeit zu evaluieren.

Die Modellleistung wird anhand des Macro-F_1-Scores für Klassifikationsaufgaben und der footprint-basierten Fitness für Generierungsaufgaben bewertet. Die Ergebnisse zeigen, dass Instruction Tuning insbesondere in den Clustern Vorhersage und Discovery die Generalisierung konsistent verbessert, insbesondere bei Modellen mit mehr als 60 Milliarden Parametern, während die Effekte bei der Anomalieerkennung weniger eindeutig ausfallen. Zusätzlich zu den quantitativen Metriken zeigt unsere qualitative Analyse, dass instruction-getunte Modelle Formatvorgaben zuverlässiger einhalten und interpretierbarere Antworten liefern als Basismodelle.

Unsere Ergebnisse belegen, dass Instruction Tuning ein vielversprechender Ansatz zur Entwicklung flexibler, semantikbewusster Modelle im Bereich Process Mining ist.

Contents

List of Tables

List of Figures

1 Introduction

This chapter sets the stage for the thesis by outlining the motivation and context behind the research. In Section 1.1, we introduce the challenges of applying Large Language Models (LLMs) to process mining, emphasizing the need for generalization across diverse tasks. Section 1.2 presents the research goals and details the main contributions of this work. Finally, Section 1.3 provides a structural overview of the thesis to guide the reader through the subsequent chapters.

1.1 Problem Statement

Process mining is a data-driven discipline that aims to extract actionable insights from event data in order to analyze, improve, and manage business processes. By leveraging event logs captured by information systems, it supports a variety of tasks such as process discovery, conformance checking, and process enhancement. However, traditional statistical and rule-based methods in process mining often fall short in capturing the semantic nuances and contextual dependencies present in real-world processes. This limits their ability to produce meaningful, flexible, and user-aligned results, especially in complex or dynamic process environments.

Recent work has introduced Natural Language Processing (NLP) techniques into process mining pipelines, enabling advances such as extracting process models from unstructured descriptions [38], detecting semantic anomalies [10], and improving event log quality through textual enrichment [36]. Building on these foundations, Large Language Models such as Llama and Mistral have emerged as powerful tools capable of encoding deeper semantic understanding and generalizing across task types. Their ability to interpret prompts and generate structured outputs makes them attractive for process-oriented analysis.

Despite their potential, existing research has largely focused on fine-tuning LLMs for specific tasks in isolation, such as anomaly detection or process discovery [35]. While these task-specific models achieve strong performance on their target tasks, they are limited in scope. They require separate training runs for each new task, depend on predefined output formats, and do not generalize well to unseen process mining tasks or variations. This not only increases the engineering effort but also hinders the scalability of LLM-based solutions.

One promising alternative, *instruction tuning*, has shown effectiveness in other domains by training models to follow natural language instructions across a variety of tasks. Yet, this approach has not been systematically explored in process mining. The absence of instruction-tuned models in this field leaves a gap in understanding whether LLMs

can generalize effectively to a wide range of process mining tasks when guided by task descriptions rather than task-specific fine-tuning.

1.2 Research Goals and Contributions

To address these limitations, this thesis investigates instruction tuning as a means of enabling general-purpose, instruction-following models for process mining. Rather than relying on task-specific fine-tuning, these models are trained to understand and execute a variety of process mining tasks based solely on natural language instructions. This approach aims to reduce the need for repeated customization, improve usability, and support generalization across unseen tasks. To explore this potential, we develop a dedicated instruction tuning framework and evaluate its performance across multiple task clusters. This leads to the central research question of this work:

Main Research Question

RQ: How effectively does instruction tuning enable large language models to generalize across diverse process mining tasks?

To systematically explore this question, we decompose it into four focused sub-questions, each addressing a key component of the instruction tuning pipeline:

RSQ1: Data Preparation
What are the essential requirements for structuring training datasets to ensure they align with the instruction format? This question guides how we curate input–output examples that are semantically meaningful and instruction-compatible.

RSQ2: Task Balancing
How can process mining tasks be grouped and balanced effectively? This addresses the challenge of ensuring representative and fair sampling across task types during model training and testing.

RSQ3: Technical Configuration
What is the optimal technical setup, including model selection and parameter configurations, for efficient instruction tuning? This includes choices around architecture, training settings, and model scale.

RSQ4: Effectiveness Assessment
To what extent does instruction tuning improve the performance of large language models on various process mining tasks? Here we evaluate whether instruction tuning leads to measurable performance improvements in both generalization and usability.

Together, these sub-questions form a comprehensive framework for assessing the potential of instruction-tuned LLMs in process mining and guide the experimental design presented in the following chapters.

To answer these questions, we conduct a series of instruction tuning experiments using open-source language models applied to a range of semantics-aware process mining tasks. Through these experiments, we evaluate how different training setups, task configurations, and model choices affect generalization and performance.

Our findings show that instruction tuning leads to promising improvements across multiple task types and demonstrate better generalization to unseen tasks. These results suggest that instruction tuning is a viable and scalable approach for developing flexible, task-agnostic models for process analysis. This work provides a foundation for future research on instruction-tuned LLMs in process analysis and contributes critical insights into how general NLP models can support domain-specific applications.

1.3 Outline

The remainder of this thesis is organized as follows:

- **Chapter 2** introduces the necessary background concepts. It begins with preliminaries on process mining and semantics-aware tasks, followed by an overview of large language models and their characteristics. The chapter concludes with a discussion of transfer learning strategies, with a focus on instruction tuning and its relevance to our work.
- **Chapter 3** reviews related literature on prompt engineering, process automation with LLMs, and benchmarking. A dedicated discussion section compares these works with our contributions and identifies gaps that this thesis aims to address.
- **Chapter 4** presents the instruction tuning methodology for process mining. It focuses on the design of the training framework, including dataset preparation, task clustering and model selection. This chapter lays the conceptual and methodological foundation for the experimental work that follows.
- **Chapter 5** presents the execution and results of the instruction tuning framework. It explains how the methodological choices from Chapter 4 were realized in practice, covering the training pipeline, datasets, model setup, and evaluation strategy. It then provides a detailed analysis of the results across tasks, domains, and output behaviors, offering evidence of the approach's effectiveness and addressing the defined research questions.
- **Chapter 6** concludes the thesis with a summary of insights, limitations of the current approach, and directions for future work. It reflects on the broader implications of the findings and outlines potential next steps for research and application.

2 Background

This chapter provides the necessary background for the concepts and methods explored in this thesis. In Section 2.1, we introduce key elements of process mining, including event data, process models, control-flow abstractions, and behavioral relations. Section 2.2 then presents the process mining tasks addressed in this work, focusing on recent semantics-aware variants. In Section 2.3, we provide an overview of LLMs, covering key NLP foundations, Transformer architecture, and practical considerations such as model size, quantization, and scalability. Finally, Section 2.4 introduces transfer learning techniques used to adapt LLMs to specific tasks, with a particular focus on instruction tuning, which plays a central role in our experimental setup.

Together, these sections establish the theoretical basis for our investigation into instruction-tuned LLMs for process mining tasks.

2.1 Process Mining Preliminaries

In this section, we provide an overview of the key artifacts and concepts used throughout this thesis in the context of process mining. We follow the original definitions introduced by Rebmann et al. [35]. These preliminaries form the foundation for the semantics-aware tasks and evaluation methods explored in later chapters.

Event Data. We adopt a simple event model, focusing on the control-flow of a process. A trace σ is a sequence that represents the events that have been recorded for the execution of a single instance of an organizational process. Such a trace consists of a finite sequence of events with each event as a record of the execution of an activity. We denote this as $\sigma = \langle a_1, ..., a_n \rangle$, with $a_i \in \mathcal{A}$, with $\mathcal{A}$ as the universe of possible activities that can be performed in organizational processes. An event log L is a finite multi-set of traces. $A_L \subset \mathcal{A}$ denotes the set of activities that appear in the traces of L.

For instance, we may have an event log $L_1 = \{\sigma_1, \sigma_2, \sigma_3\}$, with:
$\sigma_1 = \langle \textit{receive order}, \textit{accept order}, \textit{deliver package} \rangle$
$\sigma_2 = \langle \textit{receive order}, \textit{reject order} \rangle$
$\sigma_3 = \langle \textit{receive order}, \textit{deliver package} \rangle$.
This then gives $A_{L_1} = \{\textit{receive order}, \textit{accept order}, \textit{reject order}, \textit{deliver package}\}$.

Process Models. We define a process model M as the set of executions that are allowed in a process. Each execution π is represented as an activity sequence $\pi = \langle a_1, ..., a_n \rangle$, with $a_i \in \mathcal{A}$. We use $A_M \subset \mathcal{A}$, to denote the set of activities that appear in the sequences of M.

For instance, $M_1 = \{\pi_1, \pi_2\}$ with:

$\pi_1 = \langle \textit{receive order}, \textit{accept order}, \textit{deliver package} \rangle$
$\pi_2 = \langle \textit{receive order}, \textit{reject order} \rangle$
Then, $A_{M_1} = \{\textit{receive order}, \textit{accept order}, \textit{reject order}, \textit{deliver package}\}$.

Unlike process model M, which describes only valid or allowed behavior, event log L may also capture executions that deviate from the modeled process. This means that L can include traces that represent nonconformant behavior not permitted by the underlying M.

Process Trees. A process tree T is a structured, hierarchical representation of how a process can be executed. It starts from a single root node and branches out to define the control flow using operators. The leaves of the tree represent individual activities, while the internal nodes are operators that describe how their child activities or subtrees are related.

We use the following standard operators:

- $\rightarrow$ (sequence): activities are executed in order, from left to right;
- $\times$ (exclusive choice): only one of the branches is executed;
- $\wedge$ (parallel): all branches are executed, possibly in parallel;
- $\circlearrowleft$ (loop): some branches are repeated.

A process tree can be built up from:

- Single activities;
- Combinations of other trees connected by operators (like doing two activities in sequence or choosing one of several);
- Special activities like τ (tau), which means a silent step that doesn't appear in the log but may be needed for technical reasons.

For example, consider the following process tree T_{M_1}:

$$\rightarrow (\textit{receive order}, \times(\rightarrow (\textit{accept order}, \textit{deliver package}), \textit{reject order}))$$

This tree defines the following behavior: first, the order is received. Then, either the order is accepted and then the package is delivered, or the order is rejected. This corresponds to the behavior of the process model M_1 defined earlier.

Directly-Follows Relations and Graphs. The directly-follows relation $>$ captures that two activities can immediately follow each other, either in the execution sequences of a process model or the traces of an event log. For a process model M, $x > y$ if there exists an execution sequence $\pi = \langle a_1, ..., a_n \rangle \in M$ for which there exists $a_i = x, a_j = y$ with $j = i + 1$. The same applies for a log L, where these constraints should hold for at least one trace $\sigma \in L$.

We define a directly follows graph (DFG) D as a pair (A, F). A is the set of possible activities in a given process and F is a set of pairs (x, y) that represent the process' directly-follows relations (or edges in the DFG), i.e., $\{(x, y) \in A \times A \mid x > y$.

For instance, given M_1, we obtain a DFG $D_{M_1} = (A_{M_1}, F_{M_1})$, with:

$$F_{M_1} = \{(\textit{receive order}, \textit{accept order}),\ (\textit{accept order}, \textit{deliver package}),\\ (\textit{receive order}, \textit{reject order})\}$$

Directly-Follows Footprints. Following the definition of van der Aalst [40], let $D = (A, F)$ be a directly-follows graph. D defines a (directly-follows) *footprint* $fp_D : (A \times A) \to \{\to, \leftarrow, ||, \#\}$ such that for all $(x, y) \in A \times A$:

- $fp_D((x, y)) = \to$ if $(x, y) \in F$ and $(y, x) \notin F$
- $fp_D((x, y)) = \leftarrow$ if $(x, y) \notin F$ and $(y, x) \in F$
- $fp_D((x, y)) = \|$ if $(x, y) \in F$ and $(y, x) \in F$
- $fp_D((x, y)) = \#$ if $(x, y) \notin F$ and $(y, x) \notin F$

For instance, $fp_{D_{M_1}}$ is:
$fp_{D_{M_1}}((\textit{receive order}, \textit{accept order})) = \to$
$fp_{D_{M_1}}((\textit{accept order}, \textit{receive order})) = \leftarrow$
$fp_{D_{M_1}}((\textit{receive order}, \textit{reject order})) = \to$
$fp_{D_{M_1}}((\textit{reject order}, \textit{receive order})) = \leftarrow$
$fp_{D_{M_1}}((\textit{accept order}, \textit{deliver package})) = \to$
$fp_{D_{M_1}}((\textit{deliver package}, \textit{accept order})) = \leftarrow$
All other pairs in $(A_{M_1} \times A_{M_1}) = \#$

Eventually-Follows Relations. The eventually-follows relation $\prec$ is a more relaxed ordering relation than the directly-follows relation described above, focusing on activities that can either directly or indirectly follow each other in the activity sequences of a process model or in the traces of a log. For a process model M, $x \prec y$ if there exists an execution sequence $\pi = \langle a_1, ..., a_n \rangle \in M$ for which there exists $a_i = x, a_j = y$ with $i < j$. The same applies for a log L, where these constraints should hold for at least one trace $\sigma \in L$. We use EF_M to denote all eventually-follows relations of the activity sequences allowed by a model M.

For instance:

$$EF_{M_1} = \{(\textit{receive order}, \textit{accept order}),\ (\textit{receive order}, \textit{reject order}),\\ (\textit{receive order}, \textit{deliver package}),\ (\textit{accept order}, \textit{deliver package})\}$$

2.2 Process Mining Tasks

This section outlines the main types of tasks addressed in process mining. These tasks define the scope of what a system (human or model) is expected to accomplish when analyzing or interacting with process data. We begin by summarizing traditional tasks commonly found in the process mining literature, followed by a detailed explanation of semantics-aware tasks, which serve as the basis for our instruction tuning experiments. Each task is described in terms of its objective, input format, and expected output.

Traditional Tasks. Traditional process mining focuses on analyzing event logs to gain insights into how business processes are actually executed. Based on the preliminaries described in Section 2.1, we highlight several core tasks that have become foundational in the field.

Process Discovery aims to construct a process model (e.g., a process tree or Petri net) from raw event data. Given an event log $L_1 = \{\sigma_1, \sigma_2, \sigma_3\}$, the goal is to derive a process

model M that generalizes this behavior while capturing its structure. The resulting model can be visualized using directly-follows graphs or structured formalisms like process trees.

Conformance Checking compares the behavior recorded in the event log with that specified by a process model. For example, if the log contains a trace $\sigma = \langle$*receive order*, *deliver package*$\rangle$, but this sequence is not allowed by model M, the deviation indicates a conformance violation. Conformance checking helps identify non-compliant cases and is especially useful in audit and compliance contexts, for example audit trials analysis approach described by van der Aalst and de Medeiros [41].

Anomaly Detection focuses on identifying traces in the event log that deviate from the expected behavior of the process model. Unlike conformance checking, which flags all deviations, anomaly detection typically seeks unusual or rare cases that might signal fraud, errors, or inefficiencies. For instance, if $\sigma = \langle$*receive order*, *deliver package*$\rangle$ appears infrequently while not following the modeled path (which may expect acceptance or rejection), it could be flagged as anomalous.

Next Activity Prediction involves forecasting the next likely activity in an ongoing process instance. Given a partial trace such as $\langle$*receive order*, *accept order*$\rangle$, the goal is to predict that the most probable next step is *deliver package*. This task is important for proactive decision support and automation. In particular, deep learning algorithms have been increasingly utilized in the field lately [32].

Root Cause Analysis investigates the reasons behind process deviations or inefficiencies. Once an anomaly or delay is identified, root cause analysis attempts to trace it back to specific activities, decisions, or external conditions recorded in the event data.

Hypothesis Formulation allows process analysts to ask structured questions about the process, such as "Are rejected orders more likely to come from a specific region?" These hypotheses can be translated into queries over the event log and tested for patterns or correlations.

Together, these tasks form the foundation of traditional process mining and leverage the structure of event logs, process models, and behavior relations (e.g., directly-follows or eventually-follows) to extract insights about process behavior, compliance, and optimization.

Introduction to Semantics-aware Tasks. We describe five semantics-aware process mining tasks that are based on the formulation introduced by Rebmann et al. [35], with slight adaptations to fit our experimental setup. All tasks focus on the control-flow perspective and are designed so that they do not involve (nor require) access to historical event data in order to perform them. In this manner, we are able to assess whether a language model can solve the tasks based purely on its encoded knowledge of how processes generally work with respect to the meaning of activities and their inter-relations. The tasks include two forms of *semantic anomaly detection* (cf. Section 2.2.1), *semantic next activity prediction* (cf. Section 2.2.2), and two flavors of *semantic process discovery* (cf. Section 2.2.3). These tasks represent semantics-aware counterparts to well-established tasks in process mining and vary considerably in terms of their complexity.

2.2.1 Semantic Anomaly Detection

In contrast to statistical approaches that detect outliers based on the premise that infrequent behavior is anomalous [7], *semantic* anomaly detection [39] focuses on identifying process behaviors that lack logical coherence. It challenges the assumption that infrequent behavior is necessarily anomalous and emphasizes that regular occurrences do not always constitute proper process behavior. For example, from a semantic perspective, creating an invoice for a rejected purchase order constitutes anomalous behavior, regardless of how frequently it occurs.

Detecting anomalies based on process semantics requires a different approach compared to frequency-based anomaly detection. Whereas frequency-based detection can be performed by just using data in an event log (revealing statistical outliers), semantic anomaly detection requires information about how a process should (or should not) work in general. By definition, such information needs to be obtained from outside of the event log, e.g., from large knowledge bases or from the knowledge encoded in LLMs.

Two specific tasks were defined in this context, focusing on the trace and activity-relation levels.

Trace-Level Semantic Anomaly Detection. Trace-level semantic anomaly detection (T-SAD) is a binary classification problem in which a trace $\sigma \in L$ needs to be classified as anomalous or not, based on its semantics and the set of possible activities A_L, which is provided as context information. For instance, for a trace $\sigma = \langle \textit{register application}, \textit{approve application}, \textit{review application} \rangle$, the task is to classify that σ is anomalous. This is because an application should first be reviewed and only then approved (or rejected). The challenge here is that there is no specification available of the process at hand that can be used for this. Rather that anomaly needs to be inferred, requiring an understanding of how processes generally work.

Activity-Level Semantic Anomaly Detection. Activity-level semantic anomaly detection (A-SAD) is more fine-granular than T-SAD, focusing on pairs of activities in a trace rather than on an entire trace at once. In particular, A-SAD focuses on classifying any eventually-follows relation $a_i \prec_\sigma a_j$ of two activities a_i and a_j that appear in a trace $\sigma \in L$ as anomalous or not, based on its semantics and the set of possible activities A_L. For instance, given the trace $\sigma = \langle \textit{create purchase order}, \textit{reject purchase order}, \textit{create invoice} \rangle$, the eventually-follows relation *reject purchase order* $\prec_\sigma$ *create invoice* should be classified as anomalous, whereas the other pairwise relations, i.e., *create purchase order* $\prec_\sigma$ *reject purchase order* and *create purchase order* $\prec_\sigma$ *create invoice* should be classified as valid.

2.2.2 Semantic Next Activity Prediction

As a semantics-aware counterpart for next activity prediction, the semantic next activity prediction (S-NAP) task was introduced. For an incomplete trace σ, which represents an ongoing process execution in which k activities have been performed ($k \geq 1$), the task is to predict the next activity a_{k+1} in $\sigma \in L$ based on a set of possible activities A_L. For instance, given $\sigma = \langle \textit{create purchase order}, \textit{approve purchase order} \rangle$ and $A_L = \{\textit{create purchase order}, \textit{approve purchase order}, \textit{create invoice}, \textit{make payment}\}$, the task is to

predict a_{k+1} as *create invoice*. This is because, generally, an invoice should be created before a payment is made.

Whereas approaches for (traditional) next activity prediction train a model on historical traces from an event log L to predict the next activity in ongoing (i.e., unseen) executions of the process, S-NAP focuses on situations where no such historical traces are available. As a result, the next activity must be inferred by considering the semantics of the activities involved in a process.

2.2.3 Semantic Process Discovery

Semantic process discovery focuses on generating a process representation that accurately captures the underlying semantics of a process, rather than reflecting observed behavior in event logs. Unlike traditional approaches that generate models based on behavioral relations recorded from historical process executions, semantic discovery incorporates domain knowledge to ensure that the resulting models represent proper and reasonable process behavior only based on a set of possible activities. This enables the detection of not what is frequent in event data but what is meaningful and correct in a process context. Such semantic models can form the basis for downstream tasks such as conformance checking, where we can compare the appropriate behavior (captured through the semantic model), to the actually observed traces.

For instance, a semantic discovery result should represent that a *review application* activity must precede an *approve application* activity, regardless of how frequently traces in the event log adhere to or violate this rule. This is because the process representation needs to align with logical process semantics rather than reflecting patterns in historical event data. Discovering such representations requires knowledge about how processes should work, which can be sourced from domain-specific knowledge bases, human expertise or LLMs.

Two semantic discovery tasks were defined, focusing on constructing directly-follows graphs and process trees.

Semantic Directly-Follows Graph Discovery. Semantic directly-follows graph discovery (S-DFD) is the task of generating a directly-follows graph (cf. Section 2.1) that represents all reasonable relations between activities in a process. The goal is to produce a graph $D = (A, F)$, where each edge $(a, b) \in F$ reflects a valid directly-follows relation between activities a and b, only based on the set of activities A. For instance, given a set of activities {*create purchase order*, *approve purchase order*, *reject purchase order*, *create invoice*}, the semantic DFG should include the edge (*create purchase order*, *approve purchase order*). This edge aligns with the common-sense understanding that an invoice is typically created following an approved purchase order. Conversely, the DFG should not include the edge (*reject purchase order*, *create invoice*). This is because such a directly-follows relation contradicts the common-sense rule that an invoice should only be created for approved purchase orders.

The challenge lies in inferring the semantics of valid directly-follows relations without being able to rely on explicit process specifications and execution sequences.

Semantic Process Tree Discovery. Semantic process tree discovery (S-PTD) is a more structured task than S-DFD, focusing on constructing a hierarchical representation of a process, namely a process tree (cf. Section 2.1). The goal is to generate a process tree whose structure reflects the behavioral constraints in a process, such as parallelism, choices, and sequential behavior, only based on a set of possible activities A. For instance, given the activity set {*register application*, *review application*, *approve application*, *reject application*}, the semantic process tree should be $\rightarrow$(*register application*, *review application*, $\times$(*approve application*, *reject application*)). This ensures that the applications are registered before they are reviewed and, finally, a decision is made, while only one outcome per review decision should be possible.

S-PTD is particularly challenging as it requires an understanding of not only pairwise relations but also how subsets of activities (parts of the entire process) relate to one another as well as how process trees work. Doing this without having historical execution data available, necessitates external knowledge about process semantics to be able to construct trees that are both structurally and semantically sound.

2.3 Large Language Models

In this section, we provide an overview of large language models and their characteristics relevant for our research. We begin by introducing key concepts from Natural Language Processing, followed by a description of the core architecture behind modern LLMs, with a focus on the Transformers. Additionally, we highlight practical characteristics important for applying these models in our experiments, including model size, quantization, and known challenges such as computational efficiency and prompt optimization. These aspects lay the technical foundation for understanding the models used and the design decisions made throughout this thesis.

Natural Language Processing is a field that combines linguistics and machine learning to enable machines to understand and work with human language. It aims not only to comprehend individual words but also to understand their meaning in broader contexts [15].

A wide variety of tasks fall under the umbrella of NLP. Some focus on whole sentences, such as sentiment analysis or spam detection, while others involve analyzing each word to identify parts of speech or named entities. More advanced tasks include generating text from a prompt, translating or summarizing text, and extracting answers to questions based on a provided context. NLP methods also extend beyond written language, supporting challenges in speech recognition and vision language applications such as audio transcription or image captioning.

Language Models. One of the main directions in NLP is language modeling, which aims to learn the likelihood of word sequences to predict next or missing tokens (e.g., words and punctuation marks). Large Language Models are a specific category of language models that scale this concept up significantly. Rather than being trained for individual tasks, LLMs are pretrained on massive text datasets in an unsupervised manner. This contrasts

to older multitask approaches, which aimed to improve generalization by exposing models to many labeled task-specific examples [48].

Transformers. This thesis focuses on LLMs built using the transformer architecture, first introduced by Vaswani et al. [42]. Transformers are composed of layers that use a mechanism called *self-attention* to capture relationships between words in a sequence. This mechanism allows the model to "attend" to relevant words depending on the context, even if they are far apart in the text.

For example, when translating "You train this model" into German, the model must consider "You" when translating "train" and "model" when translating "this" due to grammatical dependencies like subject-verb agreement and noun gender. In contrast to this, the other words in the sentence will not matter for the translation of "model". This means that the model must sometimes attend to certain nonadjacent words, while ignoring other irrelevant words, which is made possible by self-attention [14].

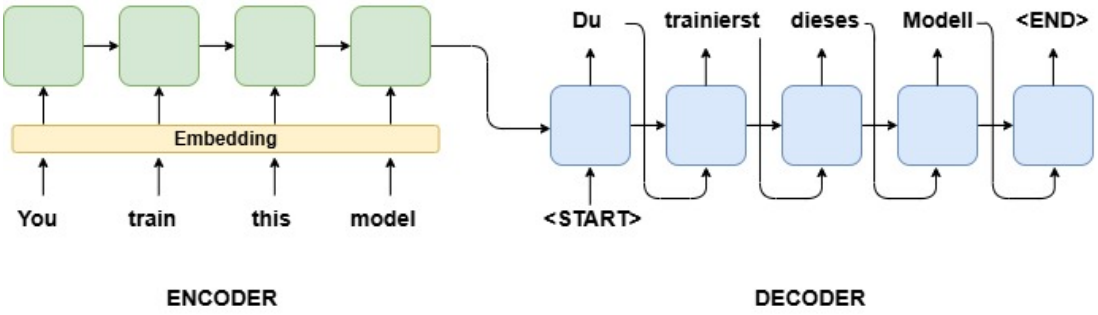

Figure 2.1: Simplified overview of the encoder-decoder architecture. Adapted from Jain et al. [22]

.

There are three primary transformer architectures: *encoder-only*, *decoder-only*, and *encoder-decoder*. *Encoder* models analyze entire input sequences and produce contextualized representations. These are bidirectional, meaning they consider all words in a sequence simultaneously, and are especially effective for *classification tasks*. The head layers of these models are often adjusted to match specific labels or desired outputs. *Decoder* models, in contrast, generate output word-by-word in a unidirectional, autoregressive manner. This means that each word is generated based on the previous words, making decoder architectures suitable for tasks such as *text generation*. In practice, they receive a prompt (essentially textual input with or without instructions), for example, the first sentence of an article we want the model to generate. The model subsequently produces the output by predicting each following sentence one token at a time.

The *encoder-decoder* models combine both mechanisms. They are typically used for sequence-to-sequence tasks where the input and output differ in structure or meaning, *translation* being a common example. One significant benefit is the independence between

the encoder and decoder components, which enables the utilization of separate models for each, thereby customizing them for specific tasks. We illustrate a simplified example of encoder-decoder usage and architecture in Figure 2.1. A sentence is first passed through the encoder component to be contextualized, and the decoder component produces an answer based on this context word by word.

To illustrate practical applications, we can look at how these architectures have been applied to process mining tasks. For example, the research conducted by Rebmann et al. [35] explores trace-level semantic anomaly detection and exemplifies both methods: (a) an encoder model is used to grasp the context and semantics of the entire trace, accompanied by an additional classifier layer that categorizes it with an appropriate label, and (b) a decoder model assesses the validity of the trace from a prompt and generatively outputs the appropriate label.

Model Size. Regardless of the architecture, transformer models can vary widely in size. The size of a model is defined by its number of *parameters* – these are the weights used in the model's internal mathematical functions to transform inputs into outputs. The architecture determines the structure of these functions (e.g., number of layers, attention heads, and hidden dimensions), while the parameters are learned during training and dictate the model's knowledge. Larger models, such as those with tens or hundreds of billions of parameters, generally have *higher capacity for learning complex patterns* and *generalizing across diverse tasks.* However, they also require significantly more *computational resources* for training and inference. Smaller models are more efficient but may struggle with more abstract or multi-step reasoning tasks. Choosing an appropriate model size thus involves a trade-off between performance and efficiency, especially in resource-constrained or domain-specific applications.

Inference. Inference refers to the process by which a model generates predictions based on input data. This occurs in two main contexts: during training, where the model uses its current parameters to make predictions and compare them with correct answers for learning, and after training, where it is deployed to generate outputs in response to new inputs. Despite its original definition, the term *inference* is often used to refer specifically to the post-training stage, when a fully trained model is used by end users to answer questions or perform tasks. In both scenarios, inference involves passing of input data through the model to produce an output, such as a label, a score, or generated text.

Quantization. Since inference is crucial during and after training, optimizing it becomes essential for large models to be efficient and deployable. One common technique for achieving this is *quantization*, which reduces memory usage and speeds up inference by converting high-precision floating-point values into smaller, lower-precision formats such as 8-bit or 4-bit integers [17]. This process is typically applied after pretraining and is known as *post-training quantization.* By compressing the model without significantly reducing accuracy, it allows large-scale language models to be used in environments with limited computational resources, such as laptops, mobile devices, or single-GPU systems.

Challenges. Despite their power, LLMs present several challenges [48]. First, training them is resource-intensive and requires high-quality data and careful parameter tuning. Prompting remains an area of active research: questions about what makes a prompt

effective, how different prompting strategies work, and how to optimize prompts for specific tasks. LLMs are also prone to hallucinations, where the model produces fluent but factually incorrect or misleading content. In some cases, they can even be manipulated to generate harmful or biased outputs, raising concerns about safety and misuse.

Finally, in Figure 2.2, we provide an overview of the open-source LLMs in the recent years, which have made powerful models accessible to the broader research community.

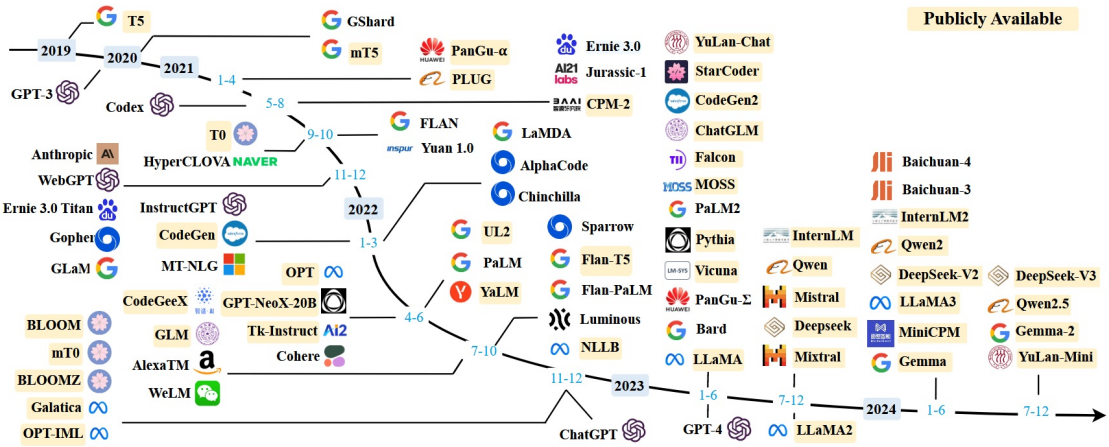

Figure 2.2: A timeline of publicly available large language models (having a size larger than 10B) in recent years by Zhao et al. [48]. Only LLMs with publicly reported evaluation results were included.

2.4 Transfer Learning and Instruction Tuning

In this section, we introduce the role of transfer learning in adapting LLMs to specialized domains such as process mining. Transfer learning allows models pre-trained on general-purpose data to be repurposed for domain-specific tasks with significantly less data and compute compared to training from scratch. We focus on two key strategies within this paradigm: in-context learning (ICL) and fine-tuning. We explain their mechanisms, advantages, and limitations, and compare them to pre-training in terms of efficiency and applicability.

Building on this foundation, we place special emphasis on instruction tuning—a supervised fine-tuning approach that teaches models to follow natural language instructions across a wide range of tasks. We discuss its benefits, practical applications, and challenges such as instruction design, consistency, and generalization. Together, these techniques form the conceptual basis for the training methodology used in our experiments.

In-Context Learning. The simplest form of adaptation is in-context learning, where no additional training is required. Instead, a few examples of input and output (also called as shots) are provided directly in the prompt together with task description. Although

the model does not genuinely learn new domain knowledge through this method, it can adjust its behavior based on the format and the limited context it sees. This method is often effective for short-term task adaptation with minimal effort, but it offers limited control or specialization. According to research conducted by Zhao et al. [48], ICL is a defining capability of modern LLMs but has limitations in depth of understanding and consistency. Additional studies have explored ways to optimize ICL with several considerations: selecting demonstrations that closely match the query [19, 25], using predefined templates [27], or trigger phrases to reveal model's step-by-step reasoning, known as chain-of-thought (CoT) prompting [46], and organizing demonstrations to prevent recency bias, where LLMs tend to replicate answers from the latter part of the demonstrations [49]. In Figure 2.3, we present a comparison of ICL and CoT. ICL involves prompting LLMs with a natural language description, multiple demonstrations, and a test query. In contrast, CoT prompting includes a sequence of intermediate reasoning steps within the prompts.

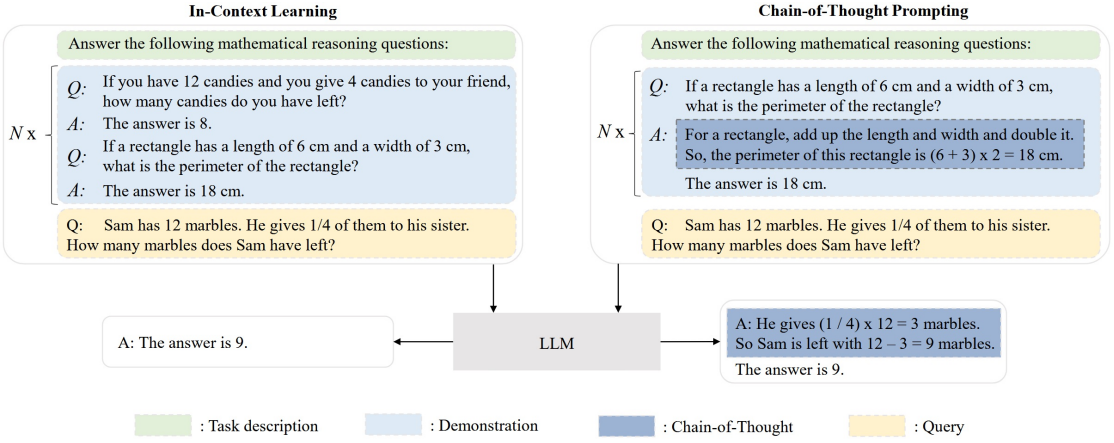

Figure 2.3: A comparative illustration of in-context learning and chain-of-thought prompting by Zhao et al. [48].

Fine-Tuning. A more effective and widely used method of adaptation is fine-tuning. This involves continuing the training of a pretrained model using a smaller, domain-specific dataset. Fine-tuning builds on existing knowledge of the model and adapts it to perform better on a specific task. This method significantly reduces the need for large datasets and computational resources compared to training from scratch. It has shown success across many NLP tasks such as text classification, sentiment analysis, and question answering [33]. In Table 2.1, we outline several aspects of pre-training versus fine-tuning to underscore the main distinctions and advantages of fine-tuning over pre-training.

There are several types of fine-tuning approaches:

- **Unsupervised fine-tuning:** This method uses unlabeled data from the target domain to refine the model's understanding of language and domain-specific vocab-

Aspect	Pre-training	Fine-tuning
Definition	Training on a vast amount of unlabeled text data	Adapting a pre-trained model to specific tasks
Data Requirement	Extensive and diverse unlabeled text data	Smaller, task-specific (usually) labeled data
Objective	Build general linguistic knowledge	Specialize model for specific tasks
Process	Data collection, training on large dataset, predict next word/sequence	Task-specific data collection, modify last layer for task, train on new dataset, generate output based on tasks
Model Modification	Entire model trained	Last layers adapted for new task
Computational Cost	High (large dataset, complex model)	Lower (smaller dataset, fine-tuning layers)
Training Duration	Weeks to months	Days to weeks
Purpose	General language understanding	Task-specific performance improvement
Examples	GPT, LLaMA 3	Fine-tuning LLaMA 3 for summarization

Table 2.1: Comparison between pre-training and fine-tuning in language model development by Parthasarathy et al. [33].

ulary. It is especially useful for niche fields like law or medicine, where labeled data is scarce. However, it lacks precision for tasks that require clearly defined output, such as classification.

- **Supervised fine-tuning (SFT):** SFT requires labeled data tailored to the target task. For example, if a model is fine-tuned to classify business data, the training data would include labeled examples that include snippets with domain language. This method yields good performance but can be expensive and time consuming due to the need for high-quality annotations.

- **Instruction tuning:** This method lies between supervised fine-tuning and prompting. It involves training the model on a variety of labeled tasks within a domain, enhanced with natural language instructions (for example, "Generate an article."), rather than just using task-specific labels and training for each task individually. The model is subsequently assessed on new, unseen tasks within the same domain, with the goal of developing a specialized assistant.

Figure 2.4 presents the key differences between instruction tuning and other model adaptation techniques like in-context learning and task-specific fine-tuning.

The Role and Challenges of Instruction Tuning. This method is especially valuable because it enables domain specialization without requiring large amounts of labeled data. In addition, it encourages the model to produce more predictable and structured responses, especially when consistent output formatting is needed. Over time, the model learns to follow patterns in how instructions are phrased and what kind of responses are expected. It is important to mention that there is still ambiguity regarding whether

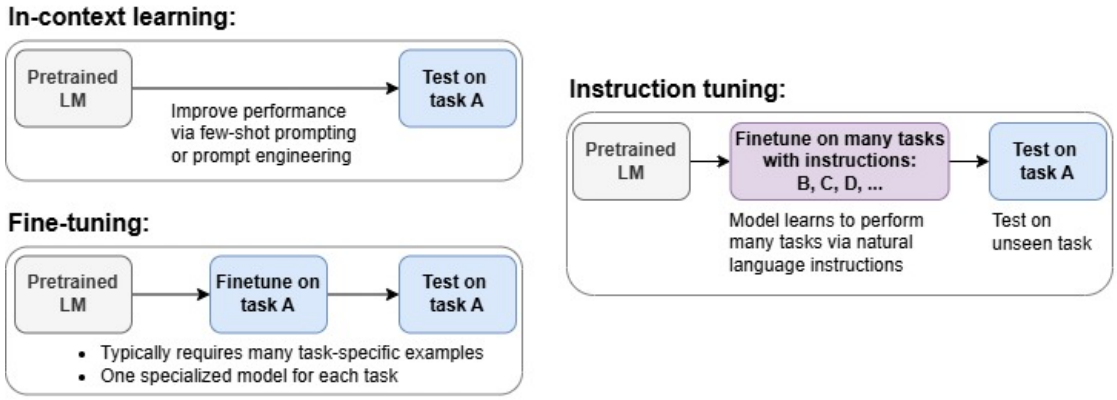

Figure 2.4: Comparison between in-context learning, fine-tuning and instruction tuning approaches of model adaptation. Adapted from Wei et al. [45].

instruction-tuned models truly grasp the semantics behind tasks or simply memorize patterns [47]. Although this issue remains under study, we continue with the assumption that instruction tuning leads to meaningful learning beyond mere memorization.

In addition to being effective for adapting LLMs to various tasks, instruction tuning also introduces several significant challenges [29]:

- **Instruction engineering:** This term is used for the process of designing task instructions. Manual (human-driven) approaches rely heavily on intuition, which can be biased, costly, and inconsistent. Attempts to automate this process, such as editing-based methods, reinforcement learning, or likelihood-based selection, often improve performance but reduce explainability. A different option is to automatically generate instructions via language models. One advantage of this method (particularly when the target and the models employed for generation are similar) is that the instruction aligns more naturally with the model. However, there is also the risk of containing irrelevant content or contradicting human reasoning, leading to a trade-off between performance and comprehensibility.

- **Instruction consistency:** Models struggle when the format of instructions changes significantly between training and evaluation, such as switching from short to long descriptions, or vice versa. This inconsistency reduces generalization and replicates the fragility seen in prompt-driven in-context learning.

- **Instruction diversity:** The practice of using different phrasings for the same task during training improves the robustness and generalization of the model. However, crafting diverse instructions manually is labor-intensive, and automatically generated alternatives often introduce noise. Despite this, combining both human and model-generated instructions can be beneficial.

- **Task and model scale:** Larger and more varied training sets typically yield better generalization, but performance gains diminish with smaller models. The advantages of instruction tuning become more consistent as both the size of the model and the variety of training tasks increase. Interestingly, the number of training samples appears to have less impact on performance than initially expected; what matters more is that these samples are well-balanced across the training dataset to prevent overfitting to dominant tasks.
- **In-context examples:** Utilizing standalone ICL techniques, like including examples alongside instructions, can further improve instruction tuning. Nevertheless, this approach faces similar difficulties to those encountered with standalone ICL. It is also important to address these challenges not only during testing but also throughout the training process. Additionally, there is a potential issue where models might depend too heavily on these examples, potentially overlooking the instructions themselves.

Despite these challenges, instruction tuning has already proven successful in several domains. Notable applications include writing [11], medical [26], and even code domains [31]. In each case, instruction-tuned models demonstrate the ability to adapt flexibly to the language and structure of domain-specific tasks.

3 Related Work

In this chapter, we focus on recent literature that uses LLMs for process-related tasks. Section 3.1 discusses the use of prompt engineering techniques to adapt LLMs to process analysis. Section 3.2 reviews approaches that employ LLMs for process automation. Section 3.3 covers advanced applications of LLMs and recent benchmarking efforts within the domain. Finally, Section 3.4 provides a comparative discussion, situating our work within the existing landscape and outlining its key contributions.

Table 3.1 provides a comparative overview of the nine works reviewed in this chapter. It summarizes the process mining tasks addressed, the types of LLMs employed, and the evaluation methods used. This comparison offers a consolidated view of how recent literature approaches the integration of language models into process analysis and where gaps still remain.

3.1 Prompt Engineering in Process Analysis

Busch et al. [8] propose prompt engineering as an alternative to fine-tuning for applying LLMs in business process management tasks, particularly in scenarios where annotated data is scarce or computational resources are limited. This study introduces a framework that leverages prompts to enable LLMs to perform tasks such as activity identification and process transformation without modifying model parameters. While the approach demonstrates promise in addressing data constraints, the authors identify several challenges, including the difficulty of process representation in prompts, the choice of a model and the processing of model's output. For instance, when extracting a process model, an additional step is required to translate the textual output of LLM into a target format.

Berti et al. [6] extend the concept of prompt engineering by addressing the context window limitations of LLMs. The authors propose abstraction techniques to compress process mining artifacts, such as event logs and process models, into textual representations that retain their semantics. Two prompting strategies are explored: (1) direct answering of predefined questions related to descriptive analysis, conformance and process improvement and (2) hypothesis formulation, where SQL queries are generated to validate hypotheses against event log databases. These techniques were implemented in the pm4py library, facilitating integration with GPT-4, and applied to case studies in domains such as healthcare, fines management and travel expense reporting. While results indicate the utility of these abstractions, the authors emphasize the need for validation across larger datasets and highlight privacy as a critical consideration.

Jessen et al. [23] investigate the application of LLMs in conversational interfaces for process mining, aiming to make the field more accessible to non-technical users. The

authors critique generic prompt engineering approaches, particularly those that directly generate SQL queries, which often leads to semantic and syntactic errors. Instead, they propose a structured agent architecture that uses role-specific prompts to provide missing context, ensuring accuracy. Evaluations are done on a corpus of process mining questions[2, 1], covering categories such as process models, event log data and advanced analysis, and demonstrate the effectiveness of the framework. Results highlight the importance of few-shot learning (including examples in prompts) and structured context in improving the relevance and comprehensibility of LLM outputs.

3.2 Process Automation with LLMs

Grohs et al. [18] evaluate the performance of LLMs, particularly GPT-4, on foundational business process management tasks, including process discovery (mining imperative and declarative models), and assess the suitability of tasks for Robotic Process Automation (RPA). The authors recommend best practices for prompt engineering, such as specifying tasks clearly, ensuring output consistency and including examples. GPT-4 demonstrates strong performance, often exceeding domain-specific solutions, indicating its versatility in addressing a range of process-related tasks with minimal configuration.

Fani et al. [16] explore the use of LLMs to automate labor-intensive tasks in RPA, such as task grouping, labeling and connector recommendations. The proposed framework leverages LLMs to improve efficiency and accuracy, outperforming traditional clustering and labeling methods. However, the study warns against fully replacing human involvement, recommending a hybrid approach that combines LLM capabilities with traditional methods. This ensures accuracy and interpretability of task groupings.

3.3 Advanced Capabilities and Benchmarking for LLMs in Process Analysis

Berti et al. [3] highlight the important capabilities of models and address the lack of comprehensive benchmarks for evaluating LLMs in process mining tasks. The authors propose evaluation frameworks focusing on key properties, including factuality, coding capabilities and accepting visual prompts, to assess LLM performance across tasks such as process discovery, conformance checking, interpreting visualizations and ensuring fairness. While existing general-purpose benchmarks provide a starting point, the authors emphasize the need for domain-specific benchmarks in process mining. Additionally, the study identifies critical implementation paradigms for effectively integrating LLMs into process mining workflows.

Building on prior work[6], other research continues investigating LLMs potentials in process mining [5]. The authors compare the performance of GPT-4 and Google's Bard on process mining tasks, often pointing out GPT-4's better output. For instance, GPT-4 demonstrates the competency across the diverse range of direct questions and better capability to formulate SQL queries. Additionally, the study explores fairness in process

mining to ensure equal treatment of different groups and transparency of a process. Using multi-prompt strategies, LLMs demonstrate the ability to provide detailed insights and ensure equitable decision-making. The findings highlight the broader potential of LLMs in advancing fairness-aware process mining.

Berti et al. [4] introduce a benchmarking framework for LLMs in process mining, including tasks such as general-purpose qualitative tasks, process model understanding, hypothesis generation (via SQL), fairness assessment and visual prompts. The authors warn against using LLMs as evaluative judges due to potential biases and emphasize the importance of ground truth in benchmarking. While GPT-4 outperforms most open-source alternatives, the study identifies scalability and accessibility challenges with smaller models, highlighting a critical area for future research.

Rebmann et al. [35] focus on fine-tuned LLMs for semantics-aware tasks such as anomaly detection, next-activity prediction and process discovery. By proposing a gold-standard benchmark, the authors avoid biases commonly associated with using LLMs for evaluation. Results indicate that fine-tuned models significantly outperform in-context learning and encoder-based methods, showcasing the importance of domain-specific training. The study suggests a hybrid approach that combines classical methods with LLM capabilities for enhanced performance.

3.4 Discussion

In the following discussion, we reflect on the integration of LLMs into process mining by outlining key benefits and highlighting current limitations. We conclude by positioning our own work within this landscape and summarizing its contributions in light of the reviewed literature.

Opportunities and Strengths of LLMs in Process Mining. The adoption of LLMs in process mining introduces a range of powerful capabilities. Most notably, these models excel at tasks requiring semantic understanding, where traditional statistical or rule-based approaches often fall short. LLMs have demonstrated strong performance in tasks such as anomaly detection, next activity prediction, and hypothesis generation, especially when nuanced interpretation of activity meaning and context is essential. Their semantic reasoning enables a more flexible and contextual analysis of process behavior, marking a shift from rigid control-flow models toward data-driven generalization.

In addition to semantic understanding, LLMs are proving valuable for generating structured outputs such as SQL queries or even process models. This facilitates the automation of tasks like discovery, query construction, and validation, and opens up new opportunities in process explanation and root cause analysis. Some studies have also explored their role in supporting fairness and identifying bias in decision-making processes.

A further advantage lies in the accessibility of LLMs for non-technical users. Through natural language interfaces and guided prompt structures, domain experts can query and analyze processes without needing deep technical knowledge. This democratization of process mining has the potential to increase adoption and reduce the dependence on specialized analysts.

Paper	Year	Approach	Process Mining Tasks Addressed	LLMs Suggested	Evaluation
Busch et al. [8]	2023	Prompt engineering for business process management	Task-agnostic concept	BERT, GPT-3, T5	-
Berti et al. [6]	2023	Abstraction techniques with prompts	Descriptive analysis, conformance checking, process improvement, hypothesis formulation	GPT-4	User-defined case study
Jessen et al. [23]	2023	Conversational agent with structured prompts	Various tasks in categories: process model, event log data, analysis, and advanced analysis	GPT-3.5-turbo, GPT-4	User-defined case study
Grohs et al. [18]	2023	Domain-specific tasks with prompt examples	Process model extraction, task classification	GPT-4	Comparison with baseline methods
Fani et al. [16]	2023	Automation of labor-intensive tasks in RPA	Task grouping, labeling, connector recommendations	GPT-3.5, GPT-4	Comparison with baseline methods
Berti et al [3]	2024	LLM capabilities and benchmarking for process mining	Process discovery, root cause analysis, anomaly detection, process improvement, interpreting visualizations, fairness	GPT-4, Google Bard / Gemini	Benchmarks (general and specific for process mining)
Berti et al. [5]	2023	Abstraction techniques with prompts an fairness analysis	Descriptive analysis, conformance checking, process improvement, hypothesis formulation, fairness evaluation	GPT-4, Google Bard	User-defined case study
Berti et al. [4]	2024	Benchmarking with diverse tasks	General-purpose qualitative tasks, process model understanding, hypothesis formulation, fairness evaluation, interpreting visualizations	GPT-4, WizardLM, Qwen2, Mixtral, Llama3	LLMs as judges
Rebmann et al. [35]	2024	Fine-tuning for sematic process mining tasks	Semantic anomaly detection, next-activity prediction, process discovery	RoBERTa, Mistral, Llama	Gold standard for semantic tasks

Table 3.1: Comparison of reviewed literature on LLMs in process mining.

Limitations and Open Challenges. However, several limitations remain. Current LLMs are constrained by a fixed context window, which restricts the amount of information they can process in a single pass. This can require simplifications or abstractions that risk omitting important behavioral details. Furthermore, LLM-generated outputs may occasionally be incomplete or incorrect, especially in tasks that require strict structural or logical accuracy. As a result, post-processing is often necessary to validate and correct generated artifacts, such as SQL queries or discovered models.

Another ongoing concern is the reliance on proprietary commercial models, which raises issues around reproducibility and accessibility in research. Many state-of-the-art results depend on models like GPT-4, which are not openly available, making it difficult to validate or build upon previous findings. Additionally, while some initial benchmarks have been proposed for evaluating LLMs on process-related tasks, they remain limited in scope. More comprehensive and domain-specific benchmarks are needed to provide a standardized basis for evaluating performance across a wider range of tasks.

Finally, there is the issue of sustainability. The high computational cost of training and

deploying LLMs, especially at large scale, raises environmental concerns [30]. As adoption grows, improving the efficiency of these models will become increasingly important to ensure their long-term viability.

Our Contribution. In light of these developments, our work makes a novel contribution by exploring whether a single, instruction-tuned model can generalize across multiple process mining tasks. Unlike prior approaches that rely heavily on task-specific fine-tuning or carefully engineered prompts, we adopt a unified instruction tuning strategy designed to teach the model a general understanding of process behavior. To ensure fair evaluation, we use a gold-standard benchmark, allowing us to measure performance without bias. Our results show that instruction tuning yields promising improvements across diverse tasks, even when those tasks were not seen during training. This suggests that it is possible to build a generalized model that understands new tasks with minimal user effort, offering both scalability and usability for real-world applications in process analysis.

4 Instruction Tuning Methodology for Process Mining

This chapter outlines the methodological design underlying our instruction tuning framework for semantics-aware process mining tasks. We focus on the preparation and structural choices that support generalization via natural language instructions, building on the background presented in Section 2.4.

We begin with the design of instructional prompts from process mining data in Section 4.1. In Section 4.2, we describe our task grouping strategy and introduce a leave-one-cluster-out setup to assess generalization to unseen task types. Finally, Section 4.3 presents our model selection criteria, including architecture, model size, and versioning.

4.1 Creation of Instruction Dataset

In this section, we describe our approach to creating a dedicated instruction dataset tailored to the process mining domain. This dataset forms the foundation for instruction tuning, enabling large language models to understand and respond to natural language instructions across a variety of tasks. We begin by outlining the main objectives and constraints, followed by the selection and adaptation of labeled data. Finally, we detail the construction of prompt templates and the generation of diverse training instances using a mix of techniques to ensure both consistency and generalization. This section directly addresses **RSQ1: Data Preparation** by focusing on the structure, variation, and format of training data.

Objectives of Dataset Creation. The first essential step in applying instruction tuning is the creation of a well-structured instruction dataset. This dataset must ensure both consistency and diversity of instructions in order to guide the model to perform various tasks in a generalizable and interpretable manner. Instruction datasets can be constructed either manually or automatically using language models. In this work, we adopt a hybrid approach: manually creating instruction examples to preserve clarity and intent, while integrating machine-generated variations derived from manual samples to enhance diversity.

Initial Dataset. A prerequisite for this step is the availability of labeled data. Ideally, one would include as many diverse task types as possible within a specific domain to maximize generalization. However, as outlined in Section 3.4, labeled datasets in the field of process mining remain limited. For this reason, we build upon the datasets introduced by Rebmann et al. [35], which cover five distinct tasks. The full details of the datasets and their structure are presented in Section 5.2.

Textual Instructions. We enrich the labeled data with instructions for each task instance. In doing so, we distinguish between three main types of textual instructions, each combining different elements of input (X), output (Y), and templates (T) [29]. We additionally illustrate these three approaches in Figure 4.1.

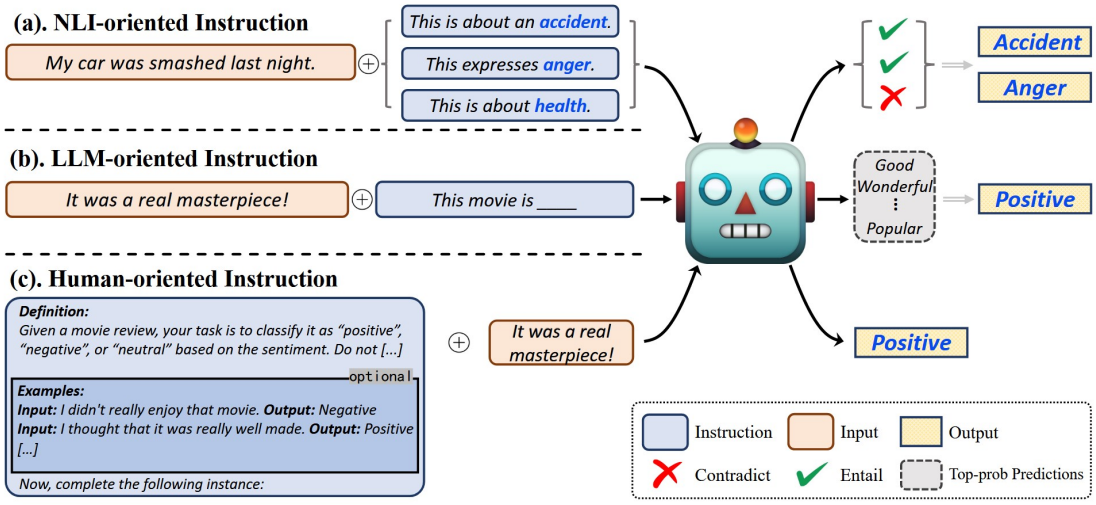

Figure 4.1: An illustration of three distinct categories of textual instructions by Lou et al. [29].

- **NLI-oriented instructions** ($I = T + Y$): These are common in classification tasks and frame the task as a natural language inference (NLI) problem. By combining templates with label semantics, the model is required to judge the validity of a hypothesis related to the label. This approach excels at classification tasks and has shown promise in zero-shot settings.
- **LLM-oriented instructions** ($I = T + X$): These include prompt-style templates that align with the pretraining objective of LLMs, such as fill-the-gap inputs. While effective and annotation-efficient, LLM-oriented instructions are often not interpretable or user-friendly. Their dependency on prompt engineering also makes them less practical in real-world applications.
- **Human-oriented instructions** ($I = T + \{X_i, Y_i\}_{i=1}^{k}$): These are designed to be understandable by non-expert users and typically take the form of descriptive, paragraph-style text. They include contextual information such as task descriptions, definitions, and specific instructions. Despite being more complex and challenging for LLMs to encode, human-oriented instructions significantly enhance generalization when used for instruction tuning.

In this thesis, we adopt human-oriented instructions, as they best align with our objective of training models to follow explicit task formulations in the process mining

domain. This fits our goal of ensuring interpretability and robustness across a range of tasks. Additionally, using such human-readable instructions enables better control over output formatting, a critical factor in evaluating the correctness of task-specific responses.

Prompt Templates. Following recent state-of-the-art instruction dataset approaches such as Flan [28] and Super-NaturalInstructions [44], we manually designed consistent prompt templates to guide model behavior. Each instruction is structured with the following components:

- **Role:** A brief statement that frames the identity of the model, typically as an intelligent assistant specialized in process mining tasks. This self-identification is intended to activate relevant domain-specific reasoning patterns and has been shown to improve model alignment with task expectations [48].
- **Task-specific instruction:** A natural language description of the task, including requirements for the expected output format. The task definitions were initially crafted by the authors of the original datasets and were subsequently refined through prior experiments to improve both their clarity and effectiveness. We adapt these definitions to align them with the goals of instruction tuning.
- **Context:** A textual representation of the input data instance (e.g., a trace or activity set) and any additional contextual information (e.g., examples) that the model must process.
- **Prompt phrase:** A closing phrase intended to naturally and seamlessly prompt the model to generate the desired result.

The correct output for each prompt is derived directly from the original dataset labels, which we reformat as natural language responses. These serve as ground-truth supervision indicators during training.

Template Diversity. To increase the robustness of the instruction dataset, we add diversity to the prompt templates by applying best practices [47]:

- **Task inversion (positive and negative):** To enrich the semantic diversity of the instruction dataset, we construct both positive and negative versions of each task where applicable. This involves flipping the relationship between input and output to reflect different task perspectives. In A-SAD and T-SAD, we prompt the model to identify either compatible or incompatible activity pairs based on the label. In S-NAP, we alternate between reconstructing a valid prefix from a final activity (positive) and correcting a faulty prefix (negative). For S-DFD, negative prompts involve intentionally flawed process graphs where the model must identify structural violations. S-PTD does not support inversion due to the complexity of the process tree format and lack of well-defined negative counterparts.
- **Prompt variations:** To improve generalization and reduce overfitting to specific phrasings, we generate multiple surface-level variants of each prompt using language

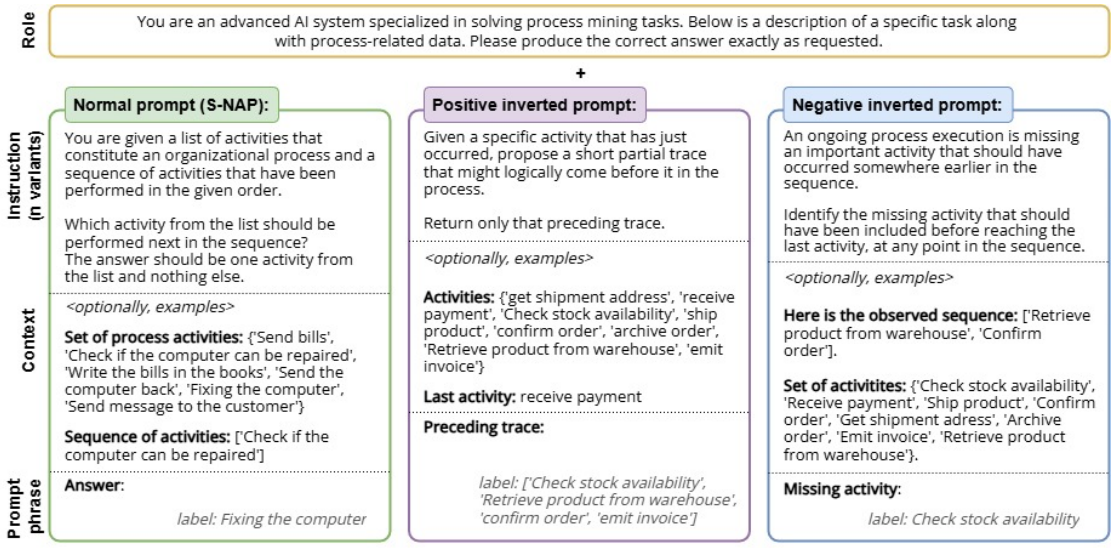

Figure 4.2: Examples of filled normal and inverted prompt templates for S-NAP task.

model paraphrasing. These variants preserve the original task semantics but differ in wording and structure. This encourages the model to rely on meaning rather than memorized patterns.

- **In-context examples (used during evaluation only):** While few-shot prompting with in-context examples can improve performance, we avoid it during training to reduce computational overhead. Instead, we apply this technique only at evaluation time, allowing us to assess its impact in a controlled manner without introducing training bias.

Techniques such as chain-of-thought prompting could further enhance reasoning capabilities. However, given the absence of process-mining-specific reasoning datasets and the increased complexity that this introduces, we do not apply such augmentations in this project.

To provide more clarity, we illustrate examples of filled normal and inverted prompt templates for the S-NAP task in Figure 4.2.

4.2 Task Clusters

In this section, we introduce our strategy for grouping process mining tasks into semantically coherent clusters to support both varied training and meaningful generalization testing. Because labeled datasets are limited, clustering helps us include a wide range of task types during training. It also enables a leave-one-cluster-out evaluation setup, where models are trained on all but one cluster and then tested on the one that was left out.

This allows us to check how well the model can handle new, unseen tasks. This approach supports **RSQ2: Task Balancing**, by making sure that the training data is diverse and that the evaluation fairly tests the model's ability to generalize.

Task Clustering for Generalization. To assess the generalization ability of an instruction-tuned model, it is essential to evaluate on tasks it has not seen during training. As discussed in previous chapters, instruction tuning aims to enable language models to follow natural language instructions across a wide variety of tasks. In this context, testing on truly unseen tasks provides a more reliable indicator of the model's transferability and robustness.

However, a major limitation in our setup is the scarcity of labeled datasets in the domain of process mining. To make the most of the available data while ensuring a meaningful generalization assessment, we group the tasks into semantically coherent *clusters*, and apply a leave-one-cluster-out training strategy. For each experiment, one cluster is reserved for evaluation, and the remaining clusters are used for training.

We define the following three task clusters based on the semantic tasks presented in Section 2.2:

- **Anomaly Detection**: includes A-SAD and T-SAD. These tasks focus on identifying whether behavior deviates from expected process execution.
- **Prediction**: includes S-NAP, where the model must determine the most likely next activity given a partial execution trace.
- **Discovery**: includes S-DFD and S-PTD, which require generating structural representations of the process.

Details of sample mixing and prompt variety for each cluster are provided in Section 5.2.

Ensuring Cluster Separation. To ensure separation between clusters, we perform a fine-grained semantic analysis of the tasks. While all tasks relate to understanding and modeling processes, they differ in their intent: anomaly detection centers on identifying deviations or abnormal behavior, prediction tasks emphasize forward-looking analysis of individual traces, and discovery tasks aim to construct abstract structural representations that capture broader process dynamics and relations. This justifies the need for three clusters. We also map the possible output formats of both original and inverted prompts for each task and remove any templates from the training data that produce output formats similar to those in the evaluation cluster[1]. This avoids the model overfitting to specific response styles and strengthens the validity of the generalization test.

4.3 Model Selection Strategy

In this section, we describe the rationale behind selecting suitable language models for instruction tuning in the process mining domain, in direct relation to **RSQ3: Technical Configuration**. We outline our decisions regarding model architecture, version,

[1]For example, we exclude an inverted prompt template for A-SAD, that returns an activity, from the training targeted at S-NAP, where expected output is also an activity.

and size. These choices are guided by practical considerations such as generative capability, training efficiency, resource constraints, and the expected impact of model scale on generalization. Together, they define the modeling setup used in our experiments.

Model Architecture. Following the fundamental concepts of LLMs outlined in Section 2.3, we conduct our analysis and develop a strategy for choosing the models that best fit our objectives. For instruction tuning, it is essential to have generative capabilities, a feature present in both encoder-decoder and decoder-only models. Since our objective is to generate prompt completions based on natural language instructions, and not to perform sequence-to-sequence transformations such as translation or summarization, *decoder-only models are a better fit.*

These models are designed to predict the next token given the previous ones in an autoregressive manner, which aligns naturally with our formulation of tasks as prompt-completion problems. To accommodate resource constraints and allow reproducibility, we select from open-source pretrained models. These models have been successfully applied in the process mining domain [35], making them a practical and tested choice.

Model Version. Large language models are often released in several versions. The most common distinction is between base models, which are pretrained on general corpora without task-specific instruction following capabilities, and instruction-tuned models, which have been further trained on a variety of tasks using natural language instructions. Given the limited scope and diversity of our available instruction tuning tasks, *we opt to use instruction-tuned versions* as a starting point.

These models already possess general instruction-following capabilities, which allow us to focus our training on adapting the model to the process mining domain and the specific task formulations. This approach also reduces training time and resource demands while still allowing us to specialize the model effectively.

Model Size. Another important factor in model selection is its size, measured by the number of parameters. In instruction tuning, larger models are generally preferred, as they are more capable of capturing diverse instruction patterns and generalizing to unseen tasks. Prior work [45] has shown that smaller models can underperform in multitask generalization due to their limited capacity.

Some recent studies, however, have shown that instruction tuning can also work well for smaller models, particularly when trained on a broad set of tasks [12]. Interestingly, improvements have been observed even with as few as nine tasks. However, in our experiments with an 8B model, we did not achieve comparable results. The model failed to generalize well to excluded tasks, likely due to the limited task variety in our setup. With only two task clusters available for training, the smaller model may have overfit to the seen tasks, leaving little capacity for transfer.

To address this, *we prioritize large-scale models* (60B+ parameters) in our experiments. This choice helps ensure sufficient capacity for generalization and aligns with our focus on evaluating instruction tuning in a low-task-diversity domain. The specific configurations are detailed in Section 5.3.1.

5 Instruction Tuning Execution and Results

This chapter presents the execution and results of our instruction-tuning framework for large language models in the process mining domain. Building on the methodological decisions outlined in Chapter 4, we describe how the framework was implemented in practice and how its performance was evaluated.

We begin in Section 5.1 with an overview of the training pipeline that guided the preparation, tuning, and evaluation phases. Section 5.2 follows with a description of the task-specific datasets and the process behavior corpus used in our experiments, including splitting and sampling strategies to ensure balanced and fair evaluation. Section 5.3 outlines the technical configuration, including model selection, training details, evaluation metrics, and hardware usage. In Section 5.4, we report and analyze the results, starting with general trends and continuing with task-specific and domain-specific performance, model behavior, few-shot evaluation, and a comparison with traditional fine-tuning.

The full source code used for dataset preparation, training, and evaluation is publicly available [1], enabling transparency and reproducibility of all experiments.

5.1 Training Pipeline

In this section, we outline the complete training pipeline used for instruction tuning in this thesis. We additionally illustrate the key steps in Figure 5.1.

The process begins with *data pre-processing*, where we clean the input datasets by removing duplicates, corrupted entries, and inconsistencies. We then apply *task balancing* to ensure that the training data is well-distributed across the clusters included in each training run. During training data preparation, we employ *dynamic prompt construction*. For each training example, we randomly select from a set of predefined prompt templates and populate these templates with relevant information from samples.

Once the instruction data is constructed, we proceed to *train a selected language model* over a specified number of epochs. Throughout training, we perform *validation* of model's performance on a held-out development subset of the unseen cluster (i.e., the cluster excluded from training). Performance is monitored using *custom metrics* tailored to the structure of each task and output format.

After training, we *select the best-performing model checkpoint* based on validation performance and *evaluate it on the full test set* of the same unseen cluster. The final model is then analyzed in terms of quantitative metrics (e.g., macro F_1-score, fitness) and

[1] `https://github.com/pirogtm7/it4pm`

through *qualitative in-depth investigation*, focusing on areas such as model robustness, error patterns, and adherence to output constraints.

This full training and evaluation cycle is repeated for each cluster, enabling us to assess how well the instruction-tuned model generalizes to new types of process mining tasks when trained on others.

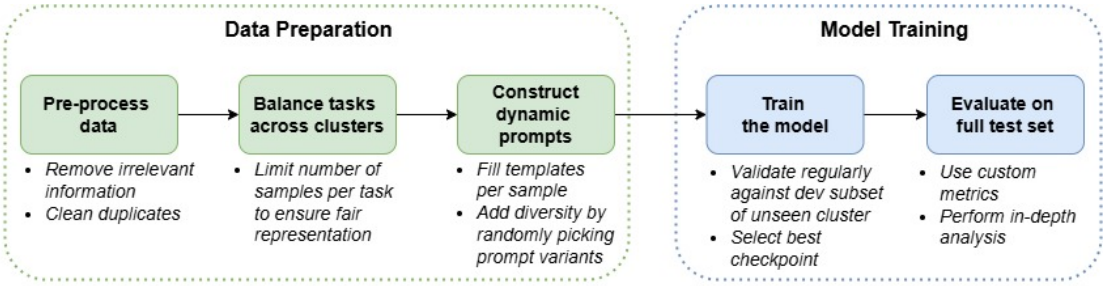

Figure 5.1: Key steps of the employed training pipeline.

5.2 Datasets

In this section, we describe the datasets used to evaluate instruction-tuned models across a variety of semantics-aware process mining tasks. The core of our evaluation is based on five task-specific datasets introduced by Rebmann et al. [35], each designed to assess a distinct capability such as anomaly detection, prediction, or discovery. These are complemented by the Process Behavior Corpus, a large and diverse collection of activity sequences derived from real BPMN models in the SAP-SAM dataset [37].

We explain how these datasets were adapted and cleaned to ensure high-quality evaluation, describe the strategy for splitting the data into training, validation, and test sets, and provide detailed statistics on sample composition. We also address **RSQ2: Task Balancing** from the perspective of sample mixing across tasks and clusters, using proportional sampling and capping strategies to ensure balanced and fair training distributions.

Finally, we discuss limitations of the dataset to contextualize the results that follow and assess how they may impact generalization to real-world process mining scenarios.

Process Behavior Corpus. Since no suitable corpus was available, Rebmann et al. created a new corpus based on graphical process models, specifically BPMN diagrams from SAP-SAM. This is a large public dataset consisting of business process models created over a decade via the academic.signavio.com platform, predominantly for research, education, and teaching purposes. To ensure high quality, only English diagrams were selected, requiring that each diagram could be transformed into a sound, block-structured workflow net. Duplicates based on identical activity sets were removed, and diagrams with fewer than two activities were excluded. For each selected workflow net, activity

sequences were generated covering all allowed behaviors, where loops are executed at most once. This ensures finite sets of traces and preserves realistic rework patterns.

We show the characteristics of the resulting corpus in Table 5.1, as defined by the authors. As depicted there, the complexity of the process models varies considerably. For instance, the median number of unique activities is 4, whereas the maximum is 21 and the process models allow for 10.34 activity sequences on average, whereas the maximum amount is 10,080.

Characteristic	**Total**	**Per process model**			
		Avg.	**Med.**	**Min.**	**Max.**
# **Process models**	15,857	–	–	–	–
# **Unique activities**	49,108	4.70	4	2	21
# **Unique sequences**	163,484	10.34	1	1	10,080

Table 5.1: Characteristics of the process behavior corpus by Rebmann et al. [35].

Dataset Splits. We adapt the original train/validation/test split strategy proposed by Rebmann et al. to ensure reproducibility. Each task-specific dataset is split based on the originating process models: 70% of models for training, 20% for validation, and 10% for testing. No activity sequence generated from a training model appears in the validation or test sets, ensuring robust evaluation without behavior leakage. Stratified sampling by the number of unique activities was used to maintain comparable complexity distributions across splits.

We additionally performed further cleaning steps to remove duplicates and irrelevant samples in each pre-defined split. We applied the following cleaning rules specific to each task:

- **T-SAD:** Ensure that combinations of unique activities and full traces are not repeated per model.
- **A-SAD:** Ensure that combinations of unique activities and activity pairs are not repeated per model. Additionally, for the validation and test splits, ensure the equal number of valid and anomalous samples.
- **S-NAP:** Remove fully completed traces and ensure that prefixes leading to the same next activity are not repeated. This preserves cases where multiple valid next activities are possible.
- **S-DFD/S-PTD:** Ensure that the set of unique activities per model is not repeated across samples.

As a result, the final number of examples per dataset differs from the original paper; we summarize the final splits in Table 5.2.

[2]When training on S-DFD and S-PTD, we combine all splits to maximize the amount of training data.

Task	Total	Train	Validation	Test
T-SAD	184,304	151,206	23,992	9,106
A-SAD	316,308	229,402	56,154	30,752
S-NAP	575,339	466,385	82,648	26,306
S-DFD	15,580	11,311[2]	2,745	1,524
S-PTD	15,580	11,311[2]	2,745	1,524

Table 5.2: Training, validation, and test split characteristics per task after cleaning original datasets introduced by Rebmann et al. [35]

We also provide additional characteristics per each task after dataset cleaning:

- **T-SAD** contains 184,304 samples. The validation and test splits are exactly balanced, while the training split includes 76,866 valid and 74,340 anomalous samples. The trace length ranges from 2 to 10 activities, with a mean of 7.26 and a median of 8.0.
- **A-SAD** comprises 316,308 samples, exactly balanced between valid (158,154) and anomalous (158,154) pairs.
- **S-NAP** includes 575,339 samples. Prefix lengths range from 1 to 9, with a mean of 5.67 and a median of 6.0.
- **S-DFD** covers 15,580 samples, with the number of edges per process graph ranging from 1 to 87. The average number of edges is 5.22, and the median is 4.0.
- **S-PTD** features 15,580 samples as well. The direct sequence operator ($\rightarrow$) is the most common, accounting for 70.15% of occurrences, followed by exclusive choice (X) at 15.78%, parallelism ($+$) at 13.29%, and loops ($*$) at 0.79%.

Sample Mixing for Training. As outlined in Section 4.2, we group tasks into clusters and train by leaving one cluster out at a time. To balance the size of different training splits during instruction tuning, we apply examples-proportional mixing [34], sampling in proportion to each task's dataset size, with an upper cap of *30,000 samples per dataset*. This selection strategy is adjusted by cluster to make full use of the available data while reducing bias towards larger datasets.

Building on the earlier prompt variation strategy, Table 5.3 shows the breakdown of prompt inversions and task contributions per training cluster. Most examples (75–85%) use the normal prompt format to stay close to the original tasks. To support generalization, 15–25% are inverted prompts, either positive (valid alternatives) or negative (anomalous cases). The exact share depends on whether a dataset allows meaningful inversion (cf. Section 4.1). Each normal prompt is drawn from six semantically similar variants, while inverted prompts have up to four variants.

Each cluster shows slightly different characteristics. The anomaly detection cluster has 61,160 samples, with 77.25% normal prompts and 14.91% negative inverted prompts to ensure anomalous examples are properly represented. Here, prediction and discovery tasks contribute equally to the training pool.

The prediction cluster contains 91,160 samples, showing the highest proportion of normal prompts at 86.84%, with small and balanced shares of positive and negative inversions. As the discovery datasets provide fewer training examples, we increase the contribution of anomaly detection tasks to make the most of their available samples, resulting in roughly a two-thirds share for the anomaly detection cluster in this training mixture.

Finally, the discovery cluster is the largest, with 120,000 examples. Here, 80% of prompts follow the normal format, with 15.04% positive and 4.96% negative inversions. In this particular cluster, we exceed the standard cap by doubling the S-NAP dataset limit to 60,000 samples, ensuring balanced representation of anomaly detection and prediction tasks during training.

Cluster	**Task**	**Samples**	**Share, (%)**	**Normal, (%)**	**Negative Inv., (%)**	**Positive Inv., (%)**
Anomaly Detection	S-NAP	30,000	49.05	31.39	9.81	7.85
	S-DFD	15,580	25.47	20.38	5.09	-
	S-PTD	15,580	25.47	25.47	-	-
	Total	**61,160**	**100.0**	**77.25**	**14.91**	**7.85**
Prediction	A-SAD	30,000	32.91	26.33	3.24	3.35
	T-SAD	30,000	32.91	26.33	3.29	3.29
	S-DFD	15,580	17.09	17.09	-	-
	S-PTD	15,580	17.09	17.09	-	-
	Total	**91,160**	**100.00**	**86.84**	**6.53**	**6.64**
Discovery	A-SAD	30,000	25.00	20.00	2.46	2.54
	T-SAD	30,000	25.00	20.00	2.50	2.50
	S-NAP	60,000	50.00	40.00	-	10.00
	Total	**120,000**	**100.00**	**80.00**	**4.96**	**15.04**

Table 5.3: Training sample distribution across clusters and tasks, including proportions of normal and inverted prompts.

Limitations. Although SAP-SAM offers the largest publicly available set of process models, it is important to acknowledge its limitations. Many models were created by students or researchers for educational purposes rather than reflecting the complexity of real-world business processes. As a result:

- Some models may emphasize technical correctness over business practicality.
- Simplifications common in academic settings (e.g., perfectly block-structured models, clear separations of phases) may not generalize fully to industry-grade processes.
- Activity labels occasionally suffer from quality issues, for example artifacts like

numbered lists (e.g., "2. Collaboration"), which might hint at structure during training, or activity sets consisting of technical commands like "git clone", which is not a traditional representation of process.

These limitations should be kept in mind when interpreting the final performance of models fine-tuned on this corpus. Nonetheless, the structured, diverse, and curated nature of the dataset still provides a strong foundation for assessing instruction-tuned LLMs in the context of process mining.

5.3 Experimental Setup

This section presents the experimental configuration used to train and evaluate instruction-tuned language models on semantics-aware process mining tasks.

Section 5.3.1 addresses **RSQ3: Technical Configuration** by introducing the large language models and baselines selected for our experiments, including both instruction-tuned and base variants. Section 5.3.2 continues this focus by describing the training procedure, hyperparameter configuration, and cluster-based tuning approach. Section 5.3.3 supports **RSQ4: Effectiveness Assessment** by detailing the evaluation metrics used for classification and generation tasks, ensuring meaningful and task-appropriate comparisons. Finally, Section 5.3.4 provides insight into the practical feasibility of instruction tuning by reporting training and inference resource usage across models.

Together, these elements define the practical setup for assessing model performance across a diverse set of process mining tasks.

5.3.1 Large Language Models / Baselines

Based on the selection criteria outlined in Section 4.3, we use two powerful decoder-only language models in our experiments: Llama 3.3 70B Instruct [3] and Mistral Large 2 Instruct (version 2407) [4]. These models were chosen due to their strong instruction-following capabilities, scale, and open-source availability. To enable efficient fine-tuning in a limited-resource environment, we use quantized 4-bit versions of both models via the Unsloth [5] framework, which provides a bridge between standard libraries like Hugging Face Transformers[6] and resource-optimized fine-tuning methods. We refer the reader to Section 2.3 for background information on quantization.

Llama 3.3 70B Instruct. This model is a multilingual, decoder-only, auto-regressive transformer optimized for text generation tasks. Its architecture follows an enhanced version of the standard transformer decoder, incorporating improvements in efficiency and alignment. Llama 3.3 was pretrained on approximately 15 trillion tokens drawn from publicly available sources, with a data cutoff in December 2023. The instruction-tuned version we use was further fine-tuned using over 25 million instruction-following examples,

[3] `https://huggingface.co/unsloth/Llama-3.3-70B-Instruct-bnb-4bit`
[4] `https://huggingface.co/unsloth/Mistral-Large-Instruct-2407-bnb-4bit`
[5] `https://github.com/unslothai/unsloth`
[6] `https://github.com/huggingface/transformers`

combining supervised fine-tuning and reinforcement learning with human feedback. The model supports multilingual text as both input and output and is designed to perform well in dialog-oriented and instruction-following tasks. Given its large parameter count and broad pretraining corpus, it is well-suited for tasks requiring semantic reasoning and generalization across domains. Its release in December 2024 makes it one of the most recent high-performing open-source models available at the time of experimentation. Llama serves as the primary model for instruction tuning in our work and is used across all experiments.

Mistral Large 2 Instruct. This model comes in a larger scale of 123 billion parameters. It is designed for long-context applications and supports a 128k token context window, making it suitable for tasks requiring extended reasoning or large input sequences. The model was released just a few months before Llama 3.3, in July 2024, so both are similar in terms of innovation. However, because Mistral is a larger model, it is expected to perform better. Mistral is instruction-tuned to follow prompts with precision and generate concise, aligned responses, even in multi-turn interactions.

Both models are first evaluated to establish baseline performance and then further instruction-tuned on our domain-specific datasets. This allows us to directly assess the impact of instruction tuning across diverse process mining tasks. Due to its higher computational requirements, we use Mistral for one additional experiment on generative tasks, while all other experiments are conducted using Llama.

5.3.2 Training Details

This section describes the fine-tuning setup for adapting large language models to process mining tasks. We follow best practices outlined in [33] and additional guidelines from the Unsloth fine-tuning documentation[7].

Efficient Training. We employ Parameter-Efficient Fine-Tuning (PEFT) strategies, specifically using QLoRA [13], an extension of LoRA [20]. *LoRA* freezes the original model weights and injects small, trainable low-rank matrices to adapt the model to new tasks. This reduces the number of updated parameters, lowers memory consumption, and improves training speed. *QLoRA* further compresses the model by quantizing the base weights to 4-bit precision while maintaining the LoRA adapters at a manageable size. This enables fine-tuning of very large models even on consumer-grade GPUs, with minimal performance loss.

We add an illustration of LoRA comparison to regular fine-tuning in Figure 5.2. In regular fine-tuning, the entire weight update matrix (ΔW) is applied to the pre-trained weights. In contrast, LoRA fine-tuning introduces two low-rank matrices (A and B) that approximate the weight update matrix (ΔW), significantly reducing the number of trainable parameters by leveraging the inner dimension (r - LoRA rank), which is a hyperparameter [33].

Prompt Masking. We use a custom data collator that masks the prompt tokens when calculating the training loss, thereby ignoring them in the loss computation. Recent

[7] `https://docs.unsloth.ai/get-started/beginner-start-here/lora-hyperparameters-guide`

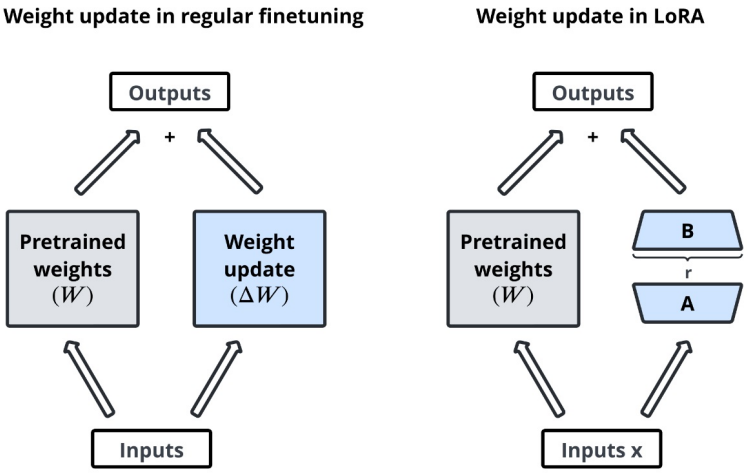

Figure 5.2: A comparison between weight updates in regular fine-tuning and LoRA fine-tuning by Parthasarathy et al. [33].

findings [43] suggest that masking the prompt speeds up model convergence and ensures the model focuses only on learning the correct output generation, rather than memorizing the repetitive structure of the instruction template, which is not the part we expect the model to recreate.

Hyperparameter Definitions. Before presenting specific values, we summarize the key hyperparameters:

- **Learning Rate:** Controls how quickly the model updates during training. Smaller learning rates make slower, more stable progress, while larger learning rates can speed up training but risk instability.
- **Batch Size:** The number of examples processed together before updating the model weights.
- **Epochs:** One full pass through the entire training dataset.
- **Optimizer:** Algorithm that adjusts the model weights to minimize loss. We use AdamW, which combines adaptive learning rates with weight decay regularization.
- **Weight Decay:** Regularization technique to prevent overfitting by slightly penalizing large weights.
- **LoRA Rank and Alpha:** Control the size and scaling of the injected low-rank matrices.

Hyperparameter Values. Our QLoRA setup operates on top of pre-quantized 4-bit models. We configure *LoRA* with a rank of $r = 16$ and a scaling factor alpha of 16 to balance model expressiveness while minimizing the risk of overfitting. We experimentally

verified that higher ranks (32 and 64) did not lead to consistent improvements. The *learning rate* is set to 1×10^{-5} and is adjusted dynamically using a linear scheduler. Training is performed with a *batch size* of 8 per device and 4 gradient accumulation steps, resulting in an effective batch size of 32. We fine-tune for a total of *3 epochs*, with a *maximum sequence length* of 1024 tokens. The *AdamW optimizer* is used with a weight decay of 0.01 to improve regularization.

Reproducibility. To minimize variance across runs, we set a fixed random seed and carefully shuffle datasets before sampling.

Training Loss. We use standard cross-entropy loss, which compares the predicted token distribution (logits) against the ground-truth labels. For example, if the true next token is "invoice" and the model predicts "payment" with higher probability, the cross-entropy penalizes the prediction based on how far off it was. Cross-entropy loss is resource-efficient and aligns well with the next-token prediction objective of language models.

Validation and Checkpoint Selection. During training, we monitor the validation performance at fixed intervals (every 250 steps) based on the dev-sets from validation splits of each task-specific dataset. Instead of using cross-entropy loss alone for model selection, we compute task-specific custom metrics better aligned with the target tasks. In cases where validation for multiple tasks must be performed (anomaly detection and discovery clusters), we average the metric improvements across tasks and select the checkpoint with the highest overall improvement.

5.3.3 Performance Metrics

We adopt the evaluation measures introduced by Rebmann et al. [35] to assess performance on both classification and discovery tasks.

Classification. We measure classification performance using the macro F_1 score. This score computes the simple average of per-class F_1 scores, giving equal weight to each class regardless of its frequency. Macro F_1 is the simple average of per-class F_1 scores, with F_1 of class c being the harmonic mean between c's precision and recall.

Generation. For discovery tasks, we evaluate generation quality using footprint-based fitness [9]. This measure compares sets of allowed execution sequences based on their pairwise behavioral footprint relations, as defined in Section 2.1. In the S-DFD task, we compute the footprint of the gold standard DFG and that of the LLM-generated DFG, then calculate fitness as the fraction of matching relations between the two. For the S-PTD task, we simulate both the gold-standard and generated process trees to obtain sets of allowed execution sequences, from which DFGs are constructed and compared in the same way as for S-DFD.

5.3.4 Hardware Usage and Execution Times

In this section, we report both training and inference (testing runs) time statistics, as well as GPU memory requirements, to contextualize the trade-offs involved in instruction tuning.

Hardware Usage. All experiments were conducted using an Nvidia A100 80GB GPU. On average, training Llama models required approximately 55 GiB of GPU memory, while inference used about 4 GiB. Mistral models were more memory-intensive, with training requiring around 78 GiB and inference approximately 66 GiB. These differences align with the increased parameter count of the Mistral model (123B) compared to Llama (70B).

Training Times. Table 5.4 presents average instruction tuning durations per epoch for each excluded cluster configuration. As expected, training times scale with the size of the training data and model. These results highlight the substantial computational cost associated with scaling up to larger model architectures, especially in multi-task learning settings.

Model	**Excluded Clusters**		
	Anomaly Detection	**Prediction**	**Discovery**
Llama (70B)	13h	18h	21h
Mistral (123B)	–	–	37h

Table 5.4: Average run times for instruction tuning with each excluded cluster (per epoch).

Inference Times. More notably, Table 5.5 shows that instruction-tuned models consistently outperform their base counterparts in terms of inference speed. For all tasks, Llama IT models complete inference significantly faster than the base models, by a factor ranging from 1.4x to over 3x. For example, in the S-DFD task, the average inference time dropped from 13.65s (Llama Base) to just 5.41s (Llama IT). A similar pattern is observed for Mistral, where IT variants halve the inference time on generation tasks. This indicates that instruction tuning is able to not only improve output quality but also enhance inference efficiency, likely because tuned models learn to produce more direct and constrained responses with less exploratory decoding.

Interestingly, these performance gains in inference occur despite IT models being exposed to a broader and more diverse prompt space during training. This suggests that instruction tuning helps models generalize more effectively and respond more confidently, reducing the likelihood of unnecessarily verbose or ambiguous outputs.

Model	**Task**				
	A-SAD	**T-SAD**	**S-NAP**	**S-DFD**	**S-PTD**
Llama Base	0.84s	0.86s	2.30s	13.65s	11.92s
Llama IT	0.62s	0.64s	1.35s	5.41s	3.86s
Mistral Base	–	–	–	16.59s	11.26s
Mistral IT	–	–	–	8.15s	6.69s

Table 5.5: Average inference times for instruction-tuned (IT) and base models (per task sample).

In summary, while instruction tuning introduces a substantial upfront training cost, especially for larger models like Mistral, it offers significant benefits in inference efficiency. These improvements are particularly valuable in real-world deployment scenarios where

response speed and resource consumption are critical. Our results suggest that with proper infrastructure and data handling, instruction-tuned models present a viable and scalable alternative to task-specific fine-tuned models.

5.4 Results and Discussion

This section presents the outcomes of our experiments and provides a comprehensive analysis of model performance across different task types, domains, and evaluation settings. It primarily addresses **RSQ4: Effectiveness Assessment**, focusing on how well instruction-tuned models generalize to unseen tasks and how different configurations impact their ability to follow process mining instructions.

We begin in Section 5.4.1 with an overview of general performance trends across all tasks and models. Section 5.4.2 dives deeper into per-task results, highlighting where instruction tuning provides the greatest gains. Section 5.4.3 explores how domain characteristics influence performance, providing insight into the models' robustness across application contexts. In Section 5.4.4, we analyze the models' adherence to instruction formats and output constraints, identifying common errors and behavior patterns. Section 5.4.5 evaluates the impact of including in-context examples during testing, while Section 5.4.6 compares our approach to traditional fine-tuning strategies, drawing on prior benchmarks to contextualize our findings.

5.4.1 General Results

We provide the results of our instruction tuning experiments across multiple process mining tasks based on test splits in Table 5.6, comparing base and instruction-tuned variants (marked as "IT") of Llama and Mistral models. Each task belongs to one of two categories: *classification* based on the macro F_1 score, and *generation* based on the fitness score.

For evaluation, we selected the following checkpoints from training: prediction-excluded at step 6000 (2.1 epochs), anomaly detection-excluded at step 5733 (3 epochs, no best-performing model identified), discovery-excluded Llama at step 8000 (2.1 epochs), and discovery-excluded Mistral at step 9000 (2.4 epochs).

Model	**Classification Tasks (macro F_1)**			**Generation Tasks (Fitness)**	
	A-SAD	**T-SAD**	**S-NAP**	**S-DFD**	**S-PTD**
Llama Base	**0.594**	**0.558**	0.525	0.630	0.621
Llama IT	0.562	0.480	**0.651**	0.714	0.697
Mistral Base	–	–	–	0.658	0.649
Mistral IT	–	–	–	**0.770**	**0.763**

Table 5.6: Performance comparison across models and tasks in instruction tuning experiments. Instruction-tuned (IT) models are compared against their base versions.

Classification Tasks. For classification tasks, we observe mixed outcomes. Instruction tuning of Llama leads to a noticeable performance gain on S-NAP, improving the F_1 score

from 0.525 to 0.651. This result indicates that instruction tuning helps the model better understand and follow instructions for predicting future process steps.

However, the instruction-tuned Llama underperforms the base version on both A-SAD and T-SAD, with F_1 drops of 3 and 8 points respectively. This suggests that instruction tuning may overwrite or dilute the model's ability to classify anomalies, especially if anomaly examples were underrepresented during fine-tuning. Another possible reason lies in the difficulty of the task itself, as assigning a label to a pair of activities or entire traces without additional justification or rich context often results in educated guessing rather than informed analysis.

Mistral model results are not available for classification tasks in this experiment, but could be investigated in future work to assess whether the performance trade-offs are model-specific or generalizable across architectures.

Generation Tasks. The results for the generation tasks clearly demonstrate the benefits of instruction tuning. For S-DFD, fitness improves from 0.630 to 0.714 with Llama, and from 0.658 to 0.770 with Mistral. Similarly, S-PTD shows an increase from 0.621 to 0.697 for Llama and from 0.649 to 0.763 for Mistral. These gains of approximately 7 to 12 points confirm that both models significantly benefit from domain-specific instruction tuning. Notably, Mistral IT outperforms Llama IT on both generation tasks, which aligns with expectations given Mistral's larger capacity.

These findings not only confirm the benefit of task-specific adaptation for structured output generation, but also suggest that instruction tuning helps models grasp higher-level process representations, particularly in cases where multiple plausible outputs exist.

5.4.2 Task-Specific Analysis

In this section, we take a closer look at the performance of instruction-tuned models across individual tasks. Every task is evaluated according to its characteristics, such as the process size (i.e., number of unique activities) and attributes particular to that task. For process size analysis, we divide the samples into groups of three activities to create balanced segments, allowing for a more consistent evaluation across varying process sizes.

To support a more nuanced interpretation of the results, we include sample count charts along with most performance plots. These indicate how many samples contributed to each score, helping to understand the stability of the result, e.g., the macro F_1 score based on 3,000 samples is more reliable than one based on only 100.

For classification tasks, we present results for Llama model only. For generation tasks, we compare performance across both Llama and Mistral models to highlight the impact of model scale and architecture on generation quality.

A-SAD

We start with analyzing the A-SAD task based on process size and prediction quality characteristics.

We first analyze the model's performance with respect to process size and prediction quality for A-SAD (Figure 5.3). We examine how the model performs across different

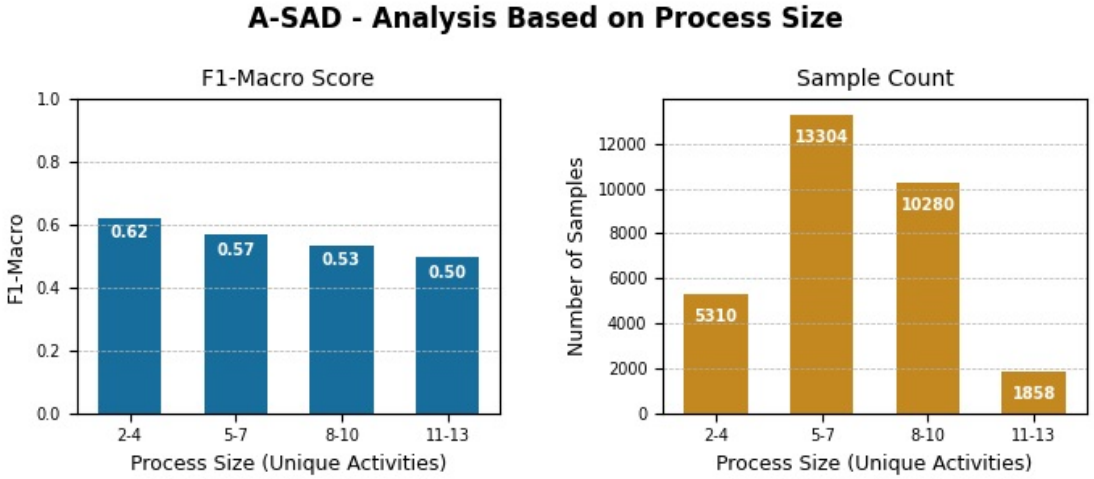

Figure 5.3: Process size based analysis for A-SAD task with Llama model. Metrics-based plot on the left is supported with sample count plot on the right.

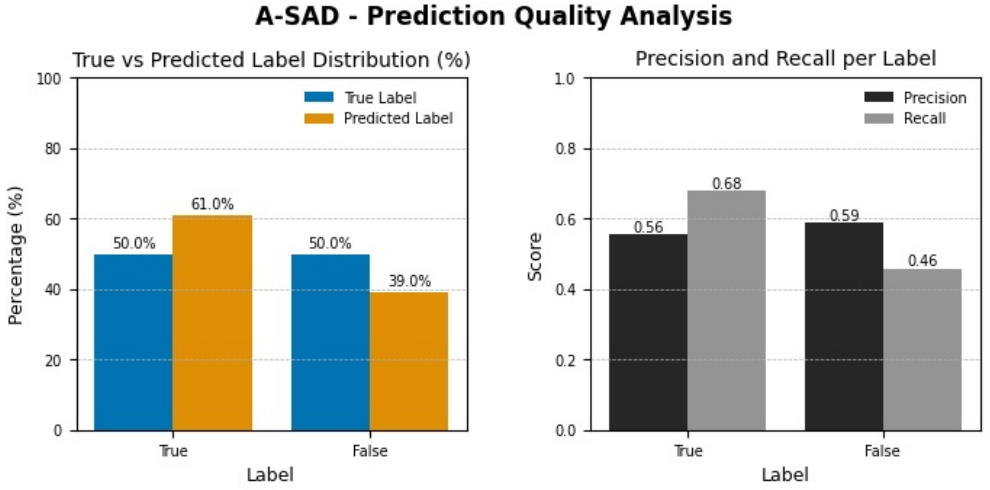

Figure 5.4: Prediction quality analysis per label for A-SAD task with Llama model. Answer share plot on the left is supported with precision vs recall plot on the right.

process sizes, categorized by the number of unique activities. We observe that as process size increases, macro F_1 score tends to decrease, with the highest performance seen in processes with 2-4 unique activities. This trend suggests that larger processes are more challenging for the model, likely due to increased complexity. However, the number of samples also plays a role, as the smaller sample size for processes with 11-13 unique

activities (only 1,858 samples) likely leads to less stable results for this group.

Next, we look into the quality of predictions, specifically focusing on the model's tendency to overpredict certain labels (Figure 5.4). The first analysis shows that the model is biased towards predicting the "True" label more frequently than "False". Despite this overprediction, the recall for the "True" label is relatively high (0.68), while the precision is lower (0.56), meaning that a significant portion of "True" instances is not being correctly assigned. For "False" labels, both precision and recall are suboptimal, with precision at 0.59 and recall at 0.46, suggesting that improvements are needed for both labels, particularly in the accurate assignment and identification of "False" instances.

T-SAD

We continue with the analysis of the T-SAD task by examining the model's behavior across different process sizes, trace lengths, and prediction quality.

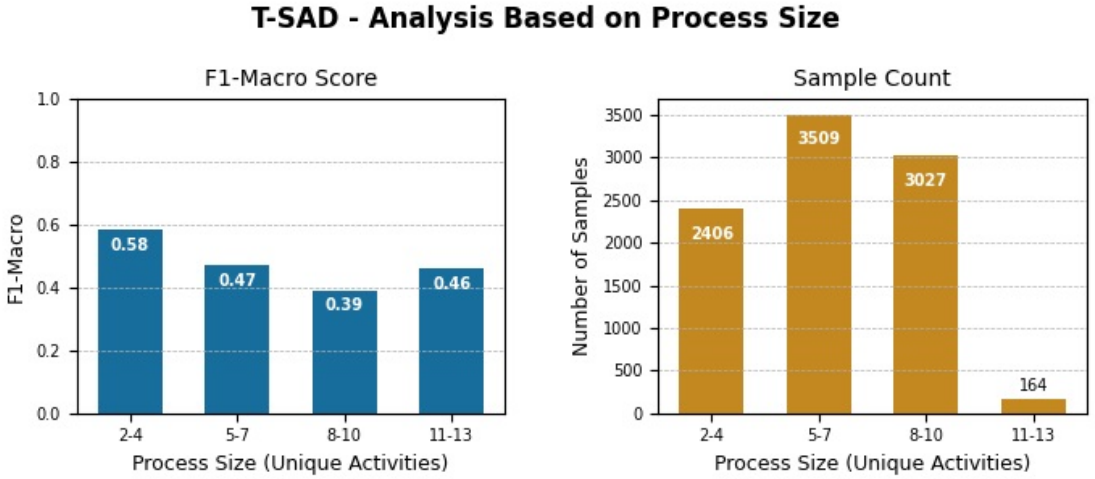

Figure 5.5: Process size based analysis for T-SAD task with Llama model. Metrics-based plot on the left is supported with sample count plot on the right.

As with A-SAD, the performance on T-SAD declines as the process size increases (Figure 5.5). Macro F_1 score is highest (0.58) for traces with 2–4 unique activities, then drops to 0.47 for sizes 5–7 and 0.39 for 8–10, before slightly increasing again to 0.46 for 11–13. This rise in the last group might seem counterintuitive but can likely be attributed to the small sample size (only 164), which makes the result less stable. In comparison to A-SAD, the performance trajectory is similar, emphasizing that process complexity from more activities negatively affects the model's ability to generalize.

In addition to process size, we also analyze how the model performs depending on the trace length, i.e., the number of activities in each trace (Figure 5.7). Macro F_1 score follows a clear downward trend: from 0.58 for short traces (2–4 activities), to 0.45 for

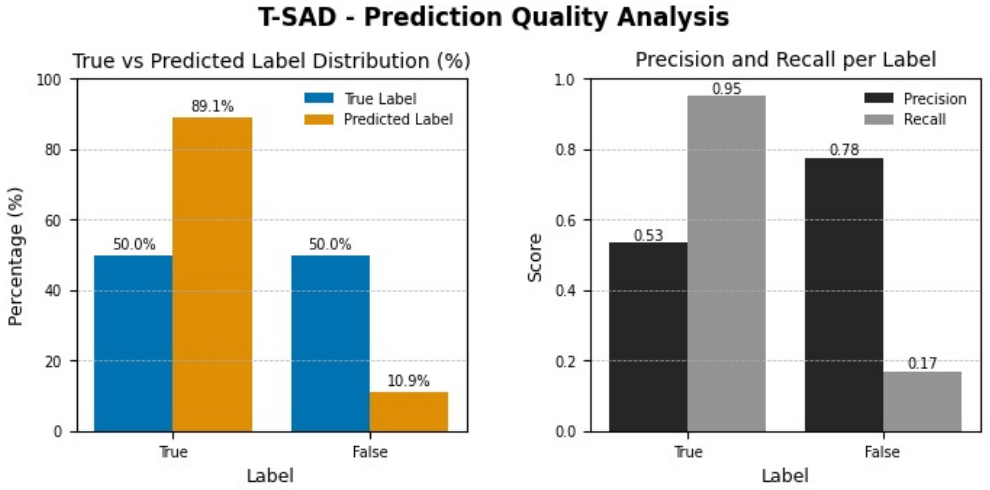

Figure 5.6: Prediction quality analysis per label for T-SAD task with Llama model. Answer share plot on the left is supported with precision vs recall plot on the right.

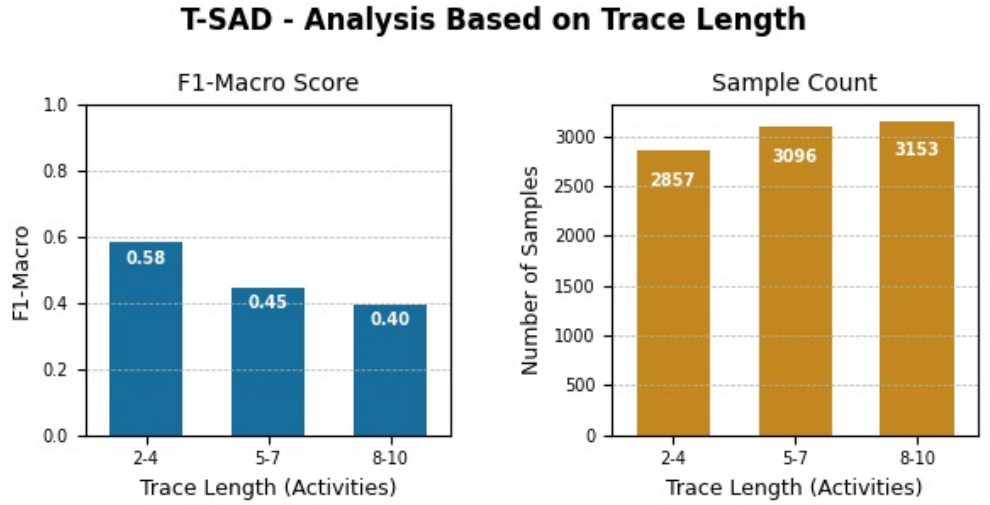

Figure 5.7: Trace length based analysis for T-SAD task with Llama model. Metrics-based plot on the left is supported with sample count plot on the right.

medium-length (5–7), and reaching the lowest value of 0.40 for the longest traces (8–10). This pattern reflects the increasing difficulty in making accurate anomaly predictions as the amount of contextual information grows, supporting the trend observed for process size. It further indicates that the model may struggle to maintain performance when handling longer sequences of behavior.

Finally, we assess prediction quality (Figure 5.6). Unlike A-SAD, where label predictions

were skewed but somewhat aligned with true label distributions, T-SAD predictions are heavily biased towards the "True" label: 89.1% of predicted labels are "True", although the actual dataset has a 50/50 distribution. This indicates an even stronger prediction bias than in A-SAD. Furthermore, while the recall for "True" is high (0.95), the precision is suboptimal (0.53), showing the model's inability to reliably assign "True" labels. The recall for "False" (0.17) is very low, showing that the model struggles to detect negative (i.e., anomalous) instances. Compared to A-SAD, these results show an even greater imbalance and confirm that classification of full traces is more challenging than reasoning about isolated activity pairs.

S-NAP

For the S-NAP task, we extend our analysis by considering two factors: the process size and the degree of prefix completion.

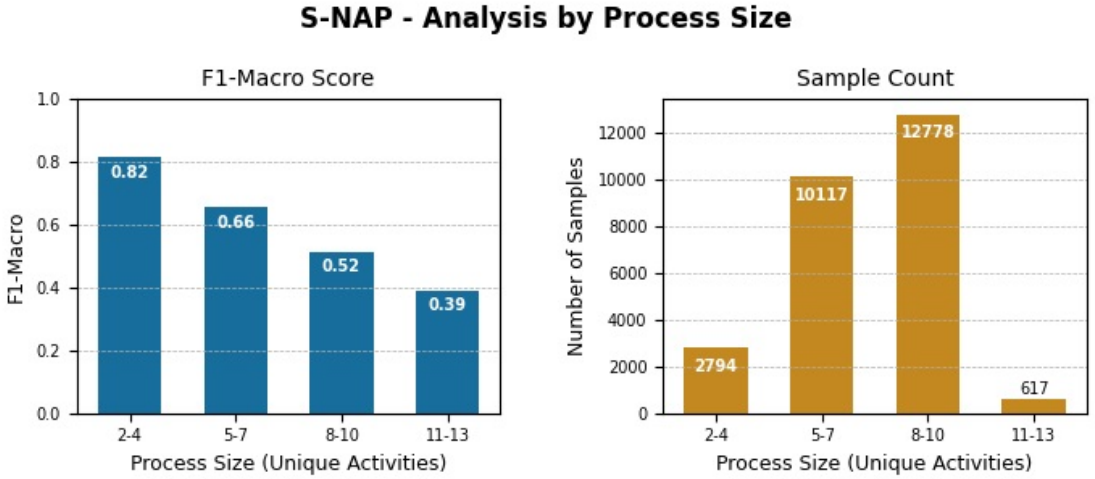

Figure 5.8: Process size based analysis for S-NAP task with Llama model. Metrics-based plot on the left is supported with sample count plot on the right.

We begin by analyzing the effect of process size (Figure 5.8). Similar to previous tasks, we observe a clear trend: as process size increases, the model's performance in terms of the macro F_1 score consistently decreases. The model performs best on small processes (2–4 unique activities) with the macro F_1 score of 0.82. This drops steadily to 0.66 for medium-small processes (5–7), 0.52 for medium-large (8–10), and reaches its lowest at 0.39 for the largest category (11–13). In the 5–10 activity number range, we notice a notable decrease by 14 points, which is validated by the significant number of samples, totaling more than 22,000. Additionally, the low number of samples in the largest category (617) likely results in less stable estimates for that group.

We next turn to the impact of prefix completion—the ratio of the prefix length to the

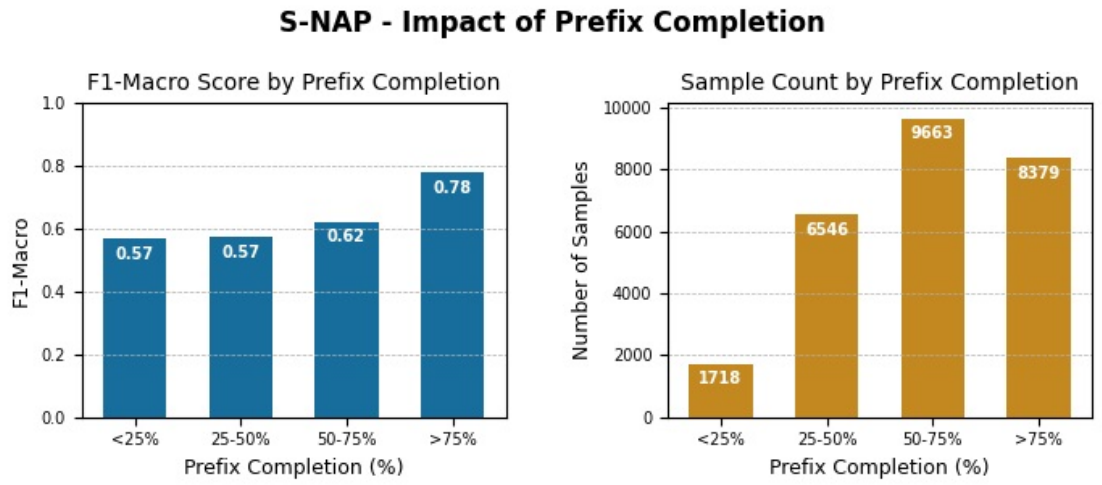

Figure 5.9: Prefix completion based analysis for S-NAP task with Llama model. Metrics-based plot on the left is supported with sample count plot on the right.

full trace length (Figure 5.9). This metric helps normalize the results across traces of different lengths by evaluating how much of the trace has already been revealed. The results show a gradual improvement in performance as more of the trace is revealed: macro F_1 starts at 0.57 for prefixes covering less than 25% of the trace, remains similar for 25–50%, slightly improves to 0.62 for 50–75%, and peaks at 0.78 for prefixes covering more than 75% of the trace. This trend is intuitive: the more of the trace is visible, the easier it becomes for the model to predict the next step.

S-DFD

We continue our task-specific analysis with S-DFD, focusing on both performance by process size and an in-depth comparison of error types produced by different models.

The first comparison (Figure 5.10) analyzes the fitness scores achieved by Llama and Mistral models across different process sizes. We observe that Mistral consistently outperforms Llama in all size categories, with the largest margin seen for processes with 8–10 unique activities (0.80 vs. 0.73). Interestingly, both models achieve stable performance across the entire spectrum, with Mistral maintaining fitness scores above 0.77 even for the largest processes. However, we should nevertheless be cautious when interpreting the results for the final group (11–13 activities), as it contains only 18 samples, making the performance estimate less reliable.

To gain more insight into the model behavior, we conduct an error type analysis (Figure 5.11). This plot shows the percentage of total samples that contain specific structural mistakes in the discovered DFGs. The most common error for both models is *Flipped Direct Order*, where the model predicts two activities in the correct pair but

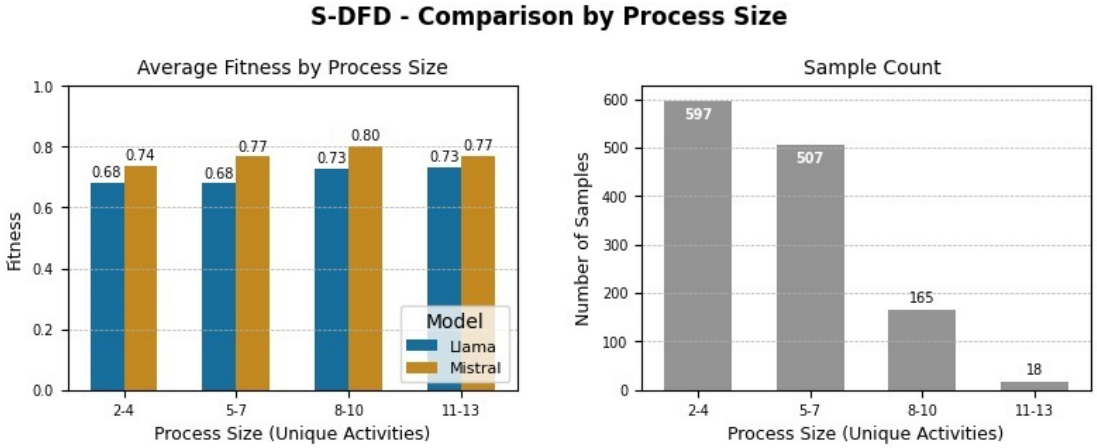

Figure 5.10: Process size based analysis for S-DFD task with comparison of Llama and Mistral models. Metrics-based plot on the left is supported with sample count plot on the right.

in the wrong direction (i.e., the predicted execution order is reversed compared to the true one). Pairs that are part of parallel behavior, where both directions are valid, are excluded from the analysis of this error type. This error occurs in over half the cases for Llama (54.7%) and is even more prevalent in Mistral outputs (62.2%), highlighting a shared struggle with correctly identifying causal direction in directly-follows relations.

The second most frequent error is *Parallelism (Missing)*, which refers to the failure to detect parallel execution patterns that exist in the ground truth. This issue is particularly evident in Mistral outputs (38.3%) compared to Llama (28.7%), suggesting that Mistral may favor overly sequential interpretations of process behavior. Conversely, *Parallelism (Extra)*—where the model incorrectly introduces concurrency between activities that should be executed sequentially—is more common in Llama (9.9%) than in Mistral (0.6%).

Another major issue is the occurrence of *Unclear End* or *Unclear Start*, where the graph structure fails to clearly indicate a proper endpoint or starting point. These are identified through the absence of nodes with exactly one incoming or outgoing edge, respectively. Mistral exhibits slightly more *Unclear End* errors (14.1%) compared to Llama (9.4%), whereas *Unclear Start* is more common in Llama (17.8%) than Mistral (10.7%).

Additional issues include *Unused Activities*, where some activities in the original set are not used in the generated model (13.1% Llama, 11.0% Mistral), and rare occurrences of *Deadlocks*, where a node is unreachable from any other, indicating a broken or isolated flow (0.4% for Llama).

Finally, the *Locked Process* error, where loops are created without any clear entry or exit, remains relatively uncommon, appearing in just 3.3% of Llama samples and 0.3% of Mistral samples.

Overall, while both models exhibit distinct strengths and weaknesses, the analysis reveals that Mistral tends to produce more structurally rigid models, sometimes at the expense of flexibility. Llama, by contrast, captures more diverse behavior but is more prone to structural noise and uncertainty.

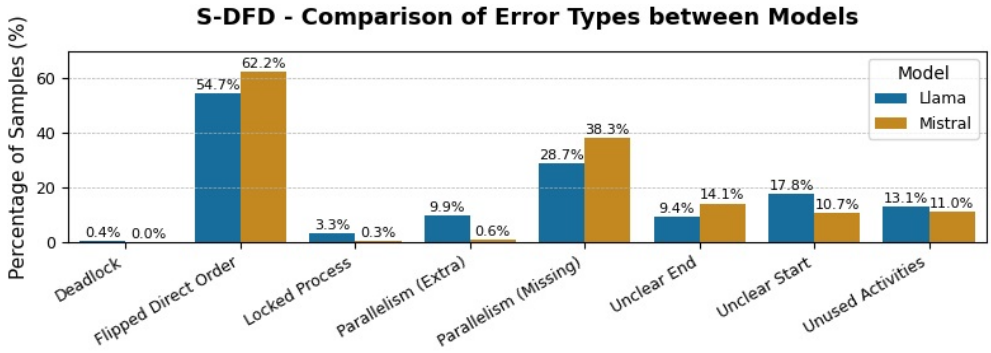

Figure 5.11: Analysis of common errors for S-DFD task with comparison of Llama and Mistral models.

S-PTD

We conclude our task-specific analysis with S-PTD, evaluating both model performance by process size and structural operator understanding through operator confusion matrices.

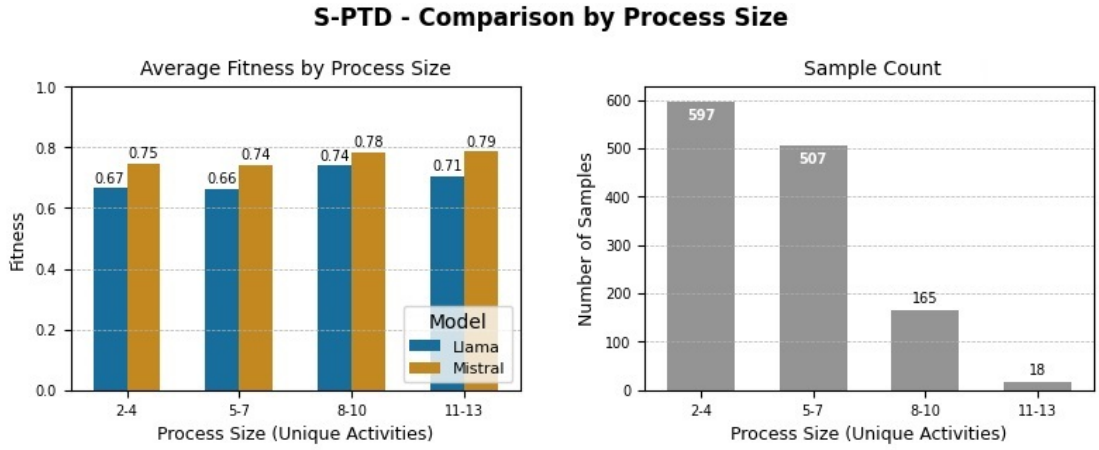

Figure 5.12: Process size based analysis for S-PTD task with comparison of Llama and Mistral models. Metrics-based plot on the left is supported with sample count plot on the right.

The process size comparison in Figure 5.12 shows consistent improvement in performance

from the Mistral model across all size categories. Fitness scores are highest for processes with 8–10 unique activities (0.78 Mistral vs. 0.74 Llama), while the lowest fitness for Llama is seen in the 5–7 range (0.66). Mistral again proves to be more robust, maintaining performance above 0.74 across all groups. Notably, for the most complex processes (11–13 activities), Mistral achieves 0.79 compared to Llama's 0.71—though this result should be interpreted with caution due to the small sample size (only 18 traces).

To understand the structural capabilities of the models in more depth, we analyze how well each model predicts the correct control-flow operators using operator confusion matrices (Figure 5.13). These matrices compare the predicted and true operators defined in Section 2.1 based on their substitution rate, i.e., how often one operator is confused with another.

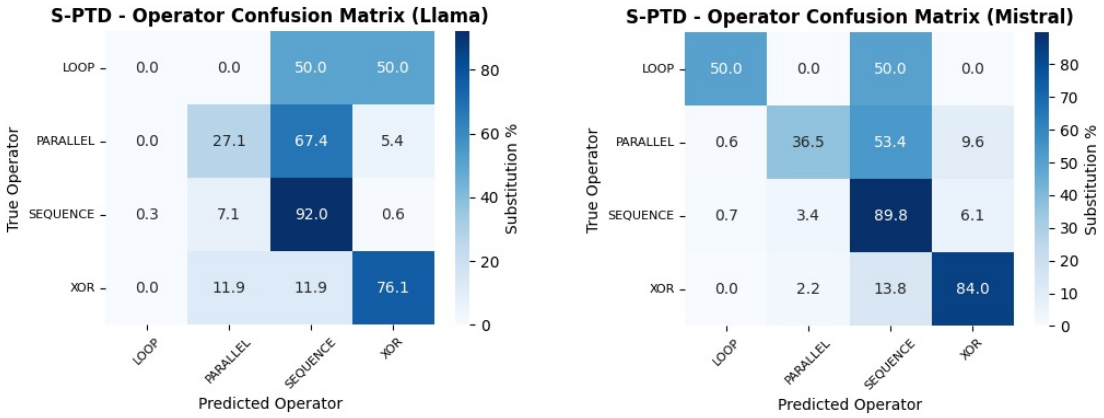

Figure 5.13: Operator confusion analysis for S-PTD task with comparison of Llama (left) and Mistral (right) models.

For *Llama (left matrix)*, the most frequently confused operator is Loop: it is never predicted correctly and is equally split between Sequence (50%) and XOR (50%). It is essential to acknowledge, however, that the loop operator is encountered in only a few samples. Parallel is also commonly misclassified, with 67.4% of parallel constructs being predicted as Sequence, and 27.1% correctly classified. In contrast, Sequence is correctly identified in 92% of cases, showing that Llama heavily favors sequential patterns. XOR also sees certain confusion with both Parallel and Sequence, each accounting for 11.9% of substitutions.

For *Mistral (right matrix)*, we observe generally stronger performance in operator classification. Sequence and XOR operators are recognized with high accuracy (89.8% and 84.0% respectively). Loop, despite being a rare structure, is split between Sequence (50%) and Loop (50%), which is an improvement over Llama. Parallel shows the most room for error: while 36.5% are correctly predicted, 53.4% are still interpreted as Sequence, confirming a shared tendency in both models to simplify concurrent structures into sequential ones.

These findings indicate that while both models struggle with accurately identifying complex control-flow operators, Mistral demonstrates greater capability overall, especially in distinguishing XOR and preserving some loop structures. Both models, meanwhile, show strong sequential bias.

The confusion matrix analysis highlights an important limitation of LLMs in structured generation tasks: while they may reproduce the overall skeleton of a process well enough to yield good fitness scores, their grasp of the finer structural semantics is still imperfect, particularly for non-sequential control constructs.

5.4.3 Domain-Specific Analysis

In this section, we investigate how the performance of instruction-tuned models varies across different business process domains. We begin by outlining the role and relevance of domains in process mining tasks, followed by an overview of the domain distribution within our datasets. We then present a detailed analysis of model performance per domain, separating the discussion into classification and generation tasks.

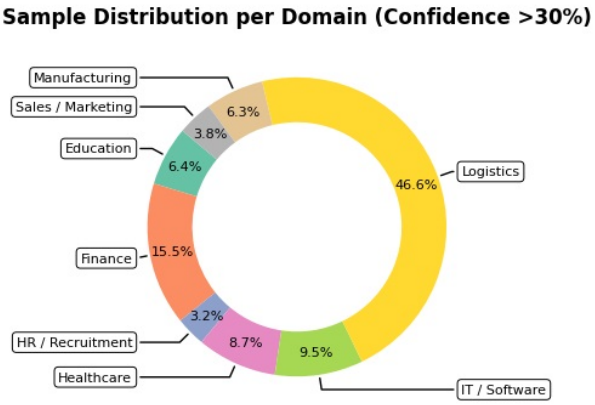

Figure 5.14: Distribution of samples considering unique process models. Only samples with confidence score 30% of the assigned label were included.

Role of Domains. In process mining, the nature of the domain can significantly influence how well models perform. Domains like healthcare, finance, or logistics vary in how structured, standardized, or flexible their processes are—factors that can affect both model generalization and the reliability of specific tasks. Moreover, many LLMs are pre-trained on corpora with uneven domain coverage, potentially favoring more common or well-documented domains like IT or finance. For these reasons, it is important to analyze how instruction-tuned models behave across different domains.

Domain Distribution. To assess this, we first assign typical domain labels to each process in the test split using a zero-shot classification model applied to the set of activities. Only domain predictions with confidence greater than 30% are considered, and each label is tied to a unique process model ID. The distribution of domains across all

samples is shown in Figure 5.14. We observe a clear dominance of the logistics domain, which accounts for 46.6% of the data, followed by finance (15.5%), IT/software (9.5%), and healthcare (8.7%). Less represented domains such as education, manufacturing, and sales/marketing make up a smaller portion of the dataset. It is worth noting that this distribution reflects unique models, while in classification tasks, individual samples are derived from these models, slightly altering the actual domain proportions in those datasets.

Classification Tasks

Next, we explore how domain-specific performance varies between classification tasks using the Llama model (Figure 5.15). Each subplot shows the macro F_1 score delta per domain relative to the task's overall score. Positive values indicate above-average performance, and negative values signal underperformance. We incorporate the precise percentage of domain samples relative to all samples in the task using "N" label and highlight domains that contribute less than 3% of the total samples in blue to caution against overinterpreting these scores due to potential instability.

For *A-SAD*, the model performs best in the logistics domain (macro F_1: 0.59), likely due to the structured and procedural nature of logistics processes, such as inventory handling, shipment routing, and order fulfillment, which tend to follow well-defined and repeatable patterns. The model also appears to develop a specific bias towards the logistics domain, since its performance diminishes in other contexts, especially domains such as IT/software (macro F_1: 0.47), sales/marketing (macro F_1: 0.49), and HR/recruitment (macro F_1: 0.51). These areas often involve more flexible workflows and higher variability in activity sequences, making it harder for the model to learn clear behavior patterns and identify anomalies based solely on activity pairs.

In the case of *T-SAD*, finance and manufacturing show the lowest performance (macro F_1: 0.44 and 0.45 accordingly), which may reflect the complexity and contextual dependence of traces in these domains. Interestingly, healthcare and education show above-average scores (macro F_1: 0.53 and 0.50 accordingly). Overall, we observe that trace-level anomaly detection is slightly more robust across domains.

Overall, the results indicate that different domains perform better in different tasks, suggesting that the model does not exhibit a consistent bias toward any single domain. While the top-performing domains vary across tasks, logistics and education stand out as more reliably aligned with the model's learned behavior, likely due to their clearer process structures or stronger representation in the training data.

In the case of *S-NAP*, the logistics domain again stands out with the highest performance (macro F_1: 0.72), while performance drops notably in domains such as IT/software and healthcare. These findings further emphasize that models generalize better in domains where task behavior is predictable and procedural.

Domain analysis reveals performance differences linked to process complexity and sample frequency, emphasizing the need for balanced domain coverage during pretraining and fine-tuning to ensure models generalize well across various contexts.

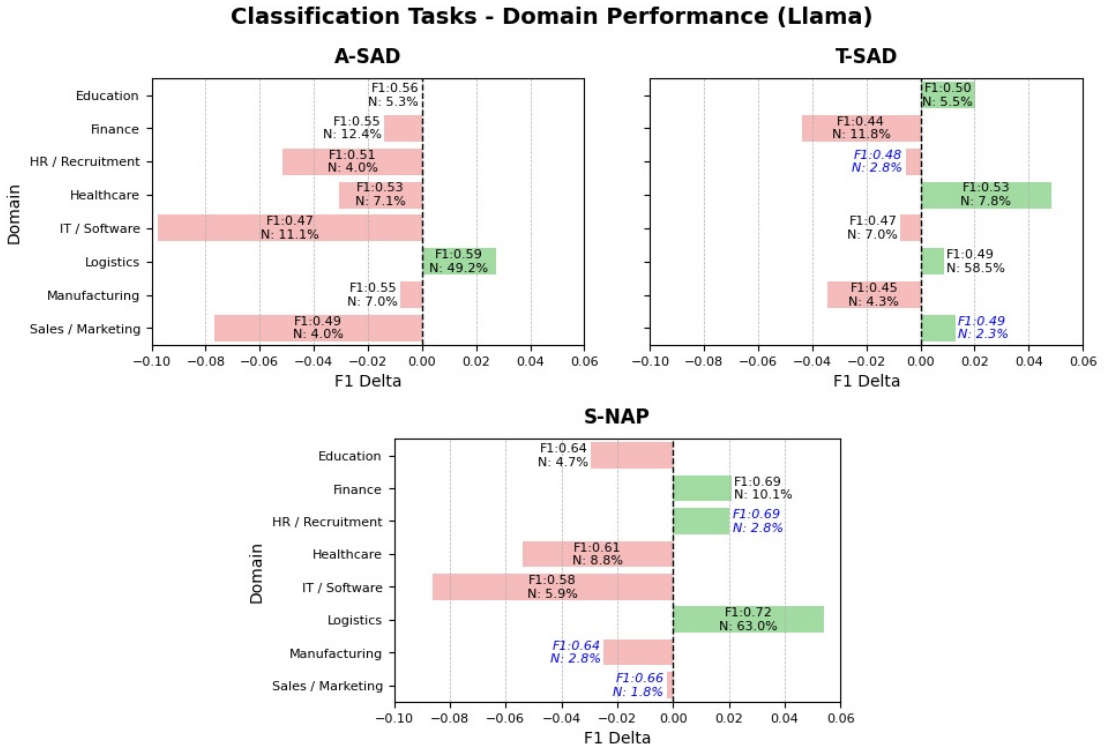

Figure 5.15: Domain performance of Llama model per A-SAD, T-SAD and S-NAP tasks, relative to average performance. Domains with representation of less than 3% of total samples are highlighted in blue.

Generation Tasks

We conclude the domain-specific analysis with the generation tasks evaluated across both Llama and Mistral models. As before, we measure domain performance by computing the fitness score delta relative to the overall average per task.

For Llama (Figure 5.16), *S-DFD* results show notable variation across domains. The model performs best in education (Fitness: 0.76) and logistics (0.74), both scoring above average. Other domains, especially healthcare and HR/recruitment, fall below the average, suggesting challenges in generalizing the DFG structure in less straightforward environments. For instance, healthcare processes often vary greatly between patients depending on their symptoms, diagnoses, and treatment paths, making them less predictable. Similarly, HR workflows, such as hiring or onboarding, are highly dependent on job role, experience level, or organizational context, and often include ad hoc decisions that are hard to capture in structured models. A similar trend appears in *S-PTD*: logistics leads with 0.71, while domains like healthcare (0.63) and sales/marketing (0.66) underperform. Interestingly, HR/recruitment, one of the least performing domains in S-DFD, delivers the

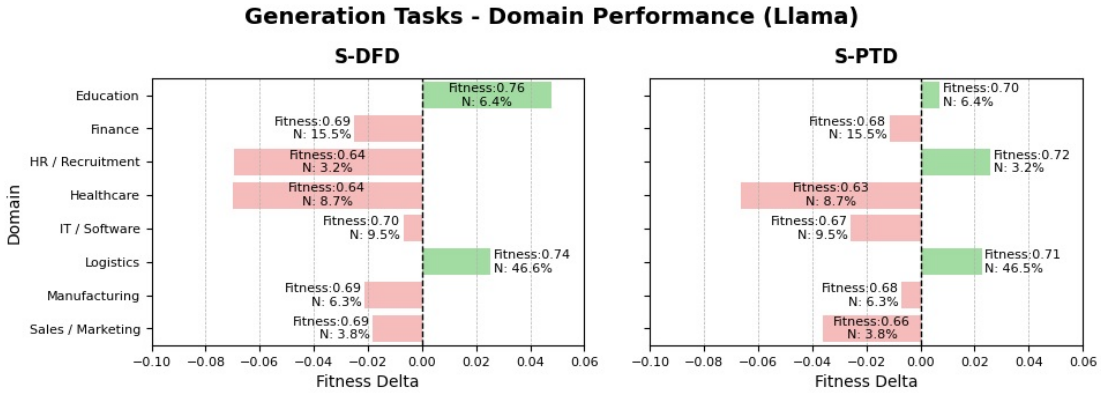

Figure 5.16: Domain performance of Llama model per S-DFD and S-PTD tasks, relative to average performance.

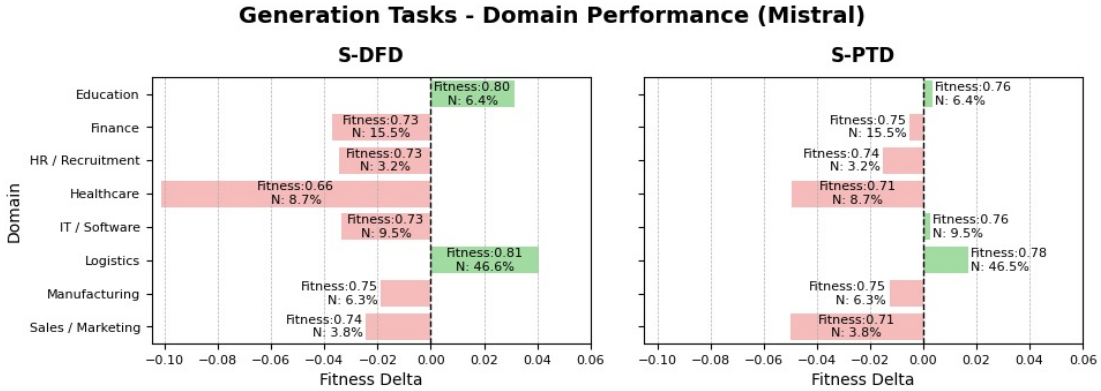

Figure 5.17: Domain performance of Mistral model per S-DFD and S-PTD tasks, relative to average performance.

highest results in S-PTD, suggesting a task-dependent sensitivity to the domain structure.

For Mistral (Figure 5.17, the differences in fitness scores are still visible. In *S-DFD*, the model performs best on education (Fitness: 0.80) and logistics (0.81), both of which likely benefit from more structured and repeatable process patterns, such as course registration workflows or shipment handling. In contrast, healthcare (0.66) is significantly below the average, indicating that Mistral may not have as much expertise in this area. In *S-PTD*, Mistral continues to perform well in logistics (0.78), while scores for healthcare (0.71) and sales/marketing (0.71) remain slightly below average. Notably, Mistral's performance in domains like finance and IT/software is close to the average, indicating that its larger capacity may help generalize across a wider range of process types. Despite this, the model still struggles slightly in domains where implicit decisions and unstructured paths

dominate.

Overall, the generation tasks confirm the earlier findings from classification: logistics and education regularly appear as the most LLM-compatible domains, benefiting from either more structured process patterns or better representation.

5.4.4 Ability to Follow Instructions

In this section, we present detailed examples to highlight specific cases and investigate how well the models follow instructions after instruction tuning. We place particular importance on separating format-related errors from semantic errors: format issues are critical to identify early, as they could influence evaluation outcomes due to parsing logic and other unforeseen factors.

Correctness of Output Formats. After instruction tuning, models mostly follow the expected output formats. For example, anomaly detection tasks consistently produce outputs as *True* or *False*. In prediction tasks, models usually return activity names, although sometimes with additional quotes or brackets, such as *['activity name']* or *'activity name'*. In discovery tasks, the output largely matches the specified format.

However, we observe several examples of incorrect output formats even after instruction tuning:

- **Empty outputs:** In some cases, the model returns no output despite a clear instruction.
- **Multiple answers in S-NAP:** Although the instruction specifies to return one activity, the model sometimes provides multiple options. For example: *['Request new employees email and accounts', 'Prepare desk and equipment for new employee', 'Create email and required accounts for new']* instead of the expected single answer: *Prepare desk and equipment for new employee*.
- **Misuse of end tokens in discovery tasks:** The special *[END]* token, intended to signal the end of an answer, is sometimes misunderstood and treated as part of the process sequence. For example: *->('settlement effected', [END])* (in S-PTD) or *Start discharging the machine -> [END]* (in S-DFD).
- **Incorrect tree structures in S-PTD:** In some cases, the generated process trees are not structured properly. For example: *->('Drop-Off', 'Data Entry') ->('Production', 'Quality Assurance') ->('Pick-Up', 'Quality Assurance') [END]*, where separators between operators are missing.

Comparison of Instruction-tuned vs Base Models. To further illustrate the improvements brought by instruction tuning, we provide a direct comparison between the base and instruction-tuned models across several representative tasks. Figures 5.18–5.21 present examples from different task types, highlighting how the tuned models follow instructions more precisely compared to their base counterparts.

In each example, the left side shows the original model input, the middle box shows the output of the base model, and the right box shows the output of the instruction-tuned model. We observe that the base models often struggle with providing properly formatted or directly relevant answers, whereas the instruction-tuned models generally adhere to the expected format and give clearer, more accurate responses. This includes better handling of output structures such as trees, activity pairs, and direct selection tasks, as well as avoiding irrelevant text generation.

Specifically, Figure 5.18 demonstrates clearer True/False responses for the T-SAD task, Figure 5.19 presents an improved next-activity prediction for the S-NAP task, Figure 5.20 highlights better formatting of activity pairs for the S-DFD task, and Figure 5.21 shows improvements in generating process trees for the S-PTD task.

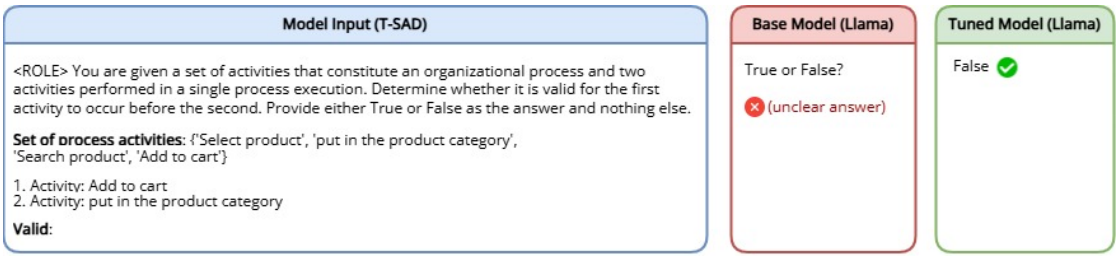

Figure 5.18: Illustration of clearer True/False responses for T-SAD task after instruction tuning.

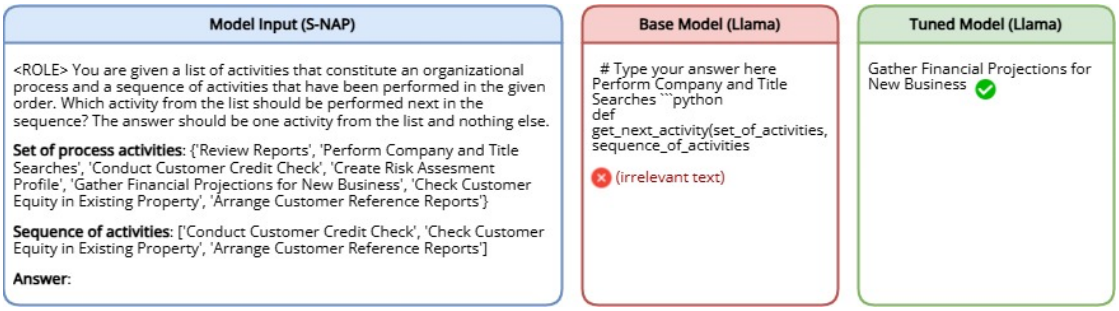

Figure 5.19: Illustration of conciseness of the answer for S-NAP task after instruction tuning.

5.4.5 Few-Shot Evaluation

In addition to standard zero-shot evaluation, we conducted experiments using few-shot prompting at test time, specifically adding 2-shot and 6-shot examples to the input prompts. Importantly, these examples were introduced only during testing—no few-shot format was used during training.

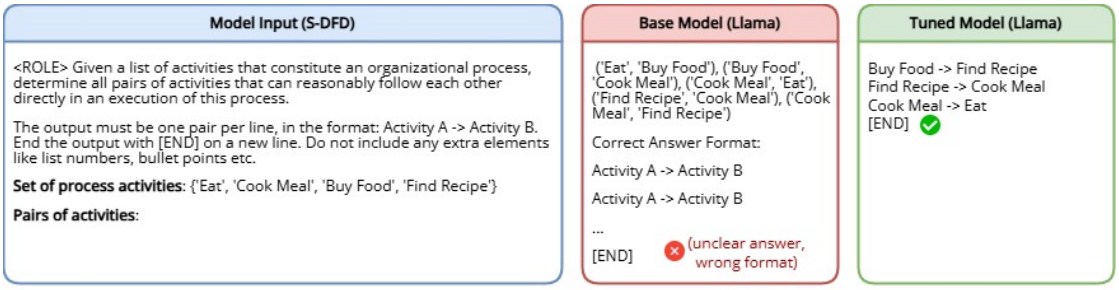

Figure 5.20: Illustration of better formatting of activity pairs for S-DFD task after instruction tuning.

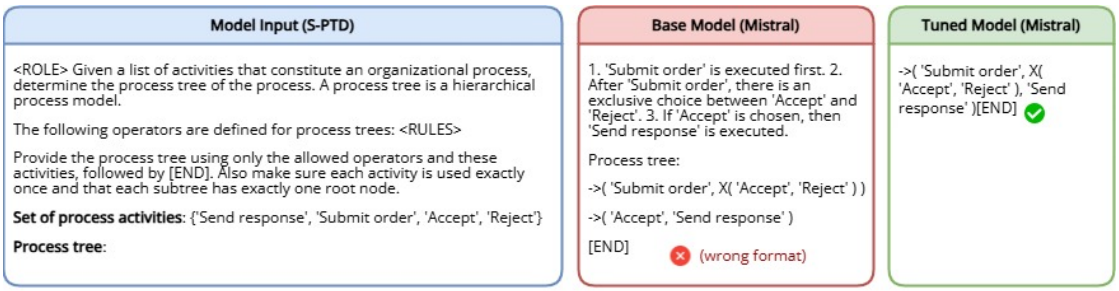

Figure 5.21: Illustration of improvements in generating process trees for S-PTD task after instruction tuning.

Contrary to expectations, this approach did not yield performance improvements. In some cases, it even resulted in a decline in model performance. Upon closer examination, we observed that adding few-shot examples often led the model to repeat the examples from the prompt, rather than generating a contextually correct answer. An illustrative example is shown in Figure 5.22, where the model, instead of producing a valid output for the current input, begins to replicate the few-shot examples and continues the pattern unnecessarily.

We hypothesize that this behavior stems from the fact that the model was never exposed to few-shot prompt structures during training, and thus lacks the ability to generalize effectively in that format. This observation aligns with findings from Longpre et al. [28], which report that training jointly with zero-shot and few-shot formats leads to improved performance.

Based on these findings, we suggest that few-shot prompting should be incorporated during training rather than introduced only at inference time. Further research is needed to explore how few-shot learning can be more effectively applied in the context of process mining tasks, particularly to avoid undesirable prompt copying and to improve

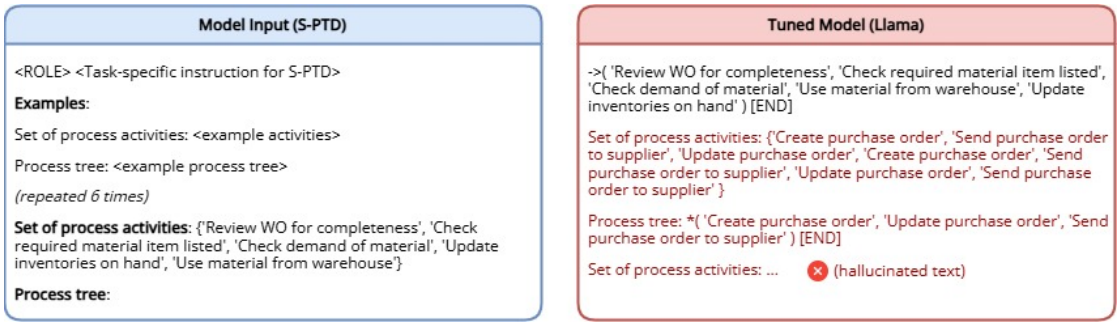

Figure 5.22: Example of model behavior during 6-shot evaluation on the S-PTD task. The model begins repeating previous examples instead of focusing on the target input.

generalization to more complex or structured outputs.

5.4.6 Instruction Tuning against Fine-Tuning

To further contextualize the results, we include a comparison against fine-tuning experiments reported in Rebmann et al. [35], where models were trained individually on each dataset and evaluated on the same dataset without the use of instruction-tuning techniques. While their models were smaller in size (7 to 8 billion parameters) than those used in this work, and therefore not directly comparable, the results provide valuable insight (cf. Table 5.7).

Notably, fine-tuning yielded improvements across all task types, including anomaly detection, suggesting that further performance gains may be possible through better task representation and data mixing strategies in instruction tuning. These findings support the idea that even general instruction tuning, when properly configured, can match or exceed the benefits of per-task fine-tuning.

Model	**Classification Tasks (macro F_1)**			**Generation Tasks (Fitness)**	
	A-SAD	**T-SAD**	**S-NAP**	**S-DFD**	**S-PTD**
Llama ICL	0.53	0.51	0.32	0.60	0.52
Llama FT	**0.88**	**0.79**	**0.69**	0.80	0.83
Mistral ICL	0.44	0.49	0.18	0.61	0.56
Mistral FT	**0.88**	**0.79**	0.68	**0.81**	**0.84**

Table 5.7: Performance comparison across models and tasks in fine-tuning experiments [35]. Fine-tuned models (FT) are trained and evaluated per task, while ICL refers to in-context learning without additional training.

6 Conclusion

This chapter concludes the thesis by summarizing the main findings and reflecting on the outcomes of our research on instruction tuning for large language models in the process mining domain. First, in Section 6.1, we review the research questions, provide consolidated answers based on experimental evidence, and highlight the key contributions of this work. Next, in Section 6.2, we reflect on the current limitations of our approach, while Section 6.3 outlines promising directions for future research.

6.1 Summary

In this thesis, we explored the application of instruction tuning to large language models in the domain of process mining. We began by providing all necessary background on process mining concepts, semantics-aware tasks, and large language models. We reviewed relevant literature, highlighting the growing importance of LLMs in process-oriented analysis and the lack of instruction tuning approaches in this context. Our contributions were positioned against this landscape and defined through a focused set of research questions.

We introduced a conceptual approach that guided the design and implementation of our experiments. This included the creation of a domain-specific instruction dataset with rich variation and controlled formatting, a task clustering strategy to enable generalization testing, careful model selection under resource constraints, and a training pipeline aligned with best practices. In the evaluation chapter, we described our datasets, experimental setup, and metrics, and provided a detailed discussion of results from multiple perspectives, including general task performance, domain variability, instruction following, and few-shot evaluation.

We now return to the research questions defined earlier and present the insights gained through our investigation.

RSQ1: Data Preparation. *What are the essential requirements for structuring training datasets to ensure they align with the instruction format?* We found that creating effective instruction datasets requires both consistency and diversity. Instructions should follow a structured prompt template including role definition, task description, input context, and a clear prompt phrase. Diversity was improved through prompt paraphrasing, inversion strategies, and balanced inclusion of anomalous cases. This design ensures generalizability while maintaining interpretability during training.

RSQ2: Task Balancing. *How can process mining tasks be grouped and balanced effectively?* To properly evaluate generalization, we grouped tasks into semantically

coherent clusters and applied a leave-one-cluster-out strategy. This ensured that models were tested only on truly unseen task types. Within each training split, we applied examples-proportional mixing with an upper cap to regulate overrepresented datasets, maximizing the use of available data while avoiding bias.

RSQ3: Technical Configuration. *What is the optimal technical setup, including model selection and parameter configurations, for efficient instruction tuning?* Effective instruction tuning requires models with sufficient capacity to generalize beyond training tasks. Our experiments showed that models below 8B parameters underperformed, while 60B+ models achieved notably better generalization. To mitigate resource limitations, we used open-source models and selected instruction-tuned checkpoints as starting points. Training strategies, such as Parameter-Efficient Fine-Tuning and careful selection of hyperparameters, were important for maintaining efficiency without compromising performance.

RSQ4: Effectiveness Assessment. *To what extent does instruction tuning improve the performance of large language models on various process mining tasks?* Evaluation across multiple tasks demonstrated that instruction tuning consistently improves model performance in most settings. Quantitatively, we observed clear improvements in two out of three task clusters:*prediction* and *discovery*. These gains were reflected in both macro-F_1 (for classification) and fitness scores (for generation). Qualitatively, instruction-tuned models produced cleaner, more structured outputs and showed greater adherence to task formats. Domain-specific analysis confirmed general robustness, with some domains (e.g., logistics and education) yielding better results than others. However, instruction tuning did not significantly benefit the *anomaly* cluster. This underperformance suggests that further investigation, particularly into dataset design and reasoning strategies, is required. Additionally, few-shot prompting during evaluation showed minimal benefits, reinforcing the importance of instruction tuning over simple in-context learning.

Main Research Question

RQ: How effectively does instruction tuning enable large language models to generalize across diverse process mining tasks?

Based on our experimental findings, instruction tuning proves to be a highly promising approach for enabling LLMs to generalize across diverse semantics-aware process mining tasks. It allows models to adapt to new task types without requiring complex prompt engineering or retraining per task. With carefully constructed instructions, balanced sampling, and well-selected model configurations, instruction-tuned models demonstrate improved generalization, and output quality. While not all task clusters benefited equally, particularly anomaly detection, our results suggest that a general-purpose, instruction-following model for process mining is both feasible and practically beneficial.

6.2 Limitations

While our work presents promising results on instruction tuning for semantics-aware process mining tasks, several scope constraints introduce opportunities for further improvement and exploration.

One major limitation is the *number of tasks used*. Our instruction tuning experiments were based on five process mining tasks, whereas other studies have shown that scaling to hundreds or thousands of tasks significantly improves generalization capabilities. As discussed in Section 2.4, the effectiveness of instruction tuning is strongly correlated with task diversity. Future research should introduce a broader set of semantics-aware tasks and datasets to fully leverage this potential and repeat key experiments under those conditions.

Another limitation is the *exclusion of advanced reasoning techniques*. In particular, we did not incorporate chain-of-thought prompting, which has shown benefits in tasks requiring complex processing. This may have contributed to the weaker performance of the anomaly cluster. Future work should explore incorporating chain-of-thought label reasoning into semantics-aware datasets to evaluate whether it improves both model accuracy and interpretability.

Additionally, while we evaluated the impact of in-context examples during testing (Section 5.4.5), we did not include such *examples during training*. Prior research suggests that joint exposure to zero-shot and few-shot formats during training can improve performance. Further studies are needed to assess the impact of including examples in the training phase for process mining tasks.

Our study also *focused exclusively on open-source models*, such as Llama and Mistral. While this ensures reproducibility and accessibility, it leaves open the question of how leading commercial models (e.g., GPT-4, Gemini) might perform on the same tasks. Evaluating state-of-the-art proprietary models could offer additional insights into performance ceilings and robustness.

From a training perspective, we used a *simple masking strategy* that completely ignored prompt tokens in the loss calculation. Recent work [21] suggests that assigning small weights to prompt tokens, rather than zero, can lead to improved performance. Future experiments should consider implementing more refined prompt loss weighting schemes.

Finally, while we evaluated models of different sizes and configurations within the discovery cluster, further experiments are needed to understand the broader *effects of model scale and architecture over other clusters*. Such experiments would deepen our understanding of how instruction tuning impacts performance across diverse settings and help refine the design of more generalizable, task-agnostic models for process mining.

6.3 Future Work

Beyond addressing the limitations outlined above, instruction tuning opens several promising directions for advancing process mining with LLMs.

First, future work should explore the applicability of instruction-tuned models to a

broader range of process mining tasks, especially those that lack well-defined benchmarks. While this thesis focused on tasks with clear evaluation standards, many real-world scenarios involve more subjective or exploratory goals (cf. Chapter 3). Investigating how instruction tuning transfers to such settings could help validate its broader utility and adaptability.

Second, instruction tuning could support the development of *general-purpose process mining assistants* that perform a variety of tasks—from answering questions and detecting anomalies to recommending improvements and generating models—all through natural language. These assistants could combine instruction tuning with techniques like retrieval-augmented generation or prompt chaining to support complex, multi-step interactions.

Third, integrating *multi-modal data*, including textual descriptions, diagrams, and execution logs, presents a promising opportunity. Instruction-tuned models that can process such diverse inputs may better reflect the complexity of real-world process environments and support more holistic analysis.

Fourth, instruction tuning can benefit from improvements in *instruction generation* itself. Automatically generating process-aware instructions using LLMs (e.g., via methods like those introduced by Li et al. [24]) could reduce the cost of dataset creation and enable more scalable training with minimal human effort.

Fifth, combining instruction-tuned LLMs with *traditional process mining techniques* offers potential for cross-perspective process intelligence. For example, frequency-based discovery methods could be complemented by LLMs capable of semantic interpretation or reasoning, improving both model accuracy and interpretability.

Finally, as instruction-tuned models move closer to deployment, *continual learning and adaptation* become critical. Exploring how instruction tuning can be extended with feedback-driven refinement or lifelong learning techniques will be key to enabling practical, robust process intelligence systems.

Bibliography

[1] Luciana Barbieri, Edmundo Madeira, Kleber Stroeh, and Wil van der Aalst. A natural language querying interface for process mining. *Journal of Intelligent Information Systems*, 61(1):113–142, 2023.

[2] Luciana Barbieri, Edmundo Roberto Mauro Madeira, Kleber Stroeh, and Wil MP van der Aalst. Towards a natural language conversational interface for process mining. In *International Conference on Process Mining*, pages 268–280. Springer International Publishing Cham, 2021.

[3] Alessandro Berti, Humam Kourani, Hannes Häfke, Chiao-Yun Li, and Daniel Schuster. Evaluating large language models in process mining: Capabilities, benchmarks, and evaluation strategies. In *International Conference on Business Process Modeling, Development and Support*, pages 13–21. Springer, 2024.

[4] Alessandro Berti, Humam Kourani, and Wil MP van der Aalst. Pm-llm-benchmark: Evaluating large language models on process mining tasks. *arXiv preprint arXiv:2407.13244*, 2024.

[5] Alessandro Berti and Mahnaz Sadat Qafari. Leveraging large language models (llms) for process mining (technical report). *arXiv preprint arXiv:2307.12701*, 2023.

[6] Alessandro Berti, Daniel Schuster, and Wil MP van der Aalst. Abstractions, scenarios, and prompt definitions for process mining with llms: a case study. In *International Conference on Business Process Management*, pages 427–439. Springer, 2023.

[7] Fábio Bezerra and Jacques Wainer. Algorithms for anomaly detection of traces in logs of process aware information systems. *Information Systems*, 38(1):33–44, 2013.

[8] Kiran Busch, Alexander Rochlitzer, Diana Sola, and Henrik Leopold. Just tell me: Prompt engineering in business process management. In *International Conference on Business Process Modeling, Development and Support*, pages 3–11. Springer, 2023.

[9] Josep Carmona, Boudewijn Van Dongen, Andreas Solti, and Matthias Weidlich. Conformance checking. *Switzerland: Springer.[Google Scholar]*, 56:12, 2018.

[10] Julian Caspary, Adrian Rebmann, and Han van der Aa. Does this make sense? machine learning-based detection of semantic anomalies in business processes. In *International Conference on Business Process Management*, pages 163–179. Springer, 2023.

[11] Tuhin Chakrabarty, Vishakh Padmakumar, and He He. Help me write a poem: Instruction tuning as a vehicle for collaborative poetry writing. *arXiv preprint arXiv:2210.13669*, 2022.

[12] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[13] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

[14] Hugging Face. How do transformers work? `https://huggingface.co/learn/llm-course/chapter1/4`, 2025. Accessed: 2025-04-20.

[15] Hugging Face. Natural language processing and large language models. `https://huggingface.co/learn/llm-course/chapter1/2`, 2025. Accessed: 2025-04-20.

[16] Mohammadreza Fani Sani, Michal Sroka, and Andrea Burattin. Llms and process mining: Challenges in rpa: Task grouping, labelling and connector recommendation. In *International Conference on Process Mining*, pages 379–391. Springer, 2023.

[17] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[18] Michael Grohs, Luka Abb, Nourhan Elsayed, and Jana-Rebecca Rehse. Large language models can accomplish business process management tasks. In *International Conference on Business Process Management*, pages 453–465. Springer, 2023.

[19] SU Hongjin, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. Selective annotation makes language models better few-shot learners. In *The Eleventh International Conference on Learning Representations*, 2022.

[20] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[21] Mathew Huerta-Enochian and Seung Yong Ko. Instruction fine-tuning: Does prompt loss matter? *arXiv preprint arXiv:2401.13586*, 2024.

[22] Aditya Jain, Gandhar Kulkarni, and Vraj Shah. Natural language processing. *International Journal of Computer Sciences and Engineering*, 6:161–167, 01 2018.

[23] Urszula Jessen, Michal Sroka, and Dirk Fahland. Chit-chat or deep talk: prompt engineering for process mining. *arXiv preprint arXiv:2307.09909*, 2023.

[24] Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason Weston, and Mike Lewis. Self-alignment with instruction backtranslation. In *ICLR*, 2024.

[25] Xiaonan Li and Xipeng Qiu. Finding supporting examples for in-context learning. *CoRR*, 2023.

[26] Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, Steve Jiang, and You Zhang. Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge. *Cureus*, 15(6), 2023.

[27] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35, 2023.

[28] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pages 22631–22648. PMLR, 2023.

[29] Renze Lou, Kai Zhang, and Wenpeng Yin. Large language model instruction following: A survey of progresses and challenges. *Computational Linguistics*, 50(3):1053–1095, 2024.

[30] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research*, 24(253):1–15, 2023.

[31] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

[32] Dominic A Neu, Johannes Lahann, and Peter Fettke. A systematic literature review on state-of-the-art deep learning methods for process prediction. *Artificial Intelligence Review*, 55(2):801–827, 2022.

[33] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296*, 2024.

[34] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[35] Adrian Rebmann, Fabian David Schmidt, Goran Glavaš, and Han van Der Aa. Evaluating the ability of llms to solve semantics-aware process mining tasks. In *2024 6th International Conference on Process Mining (ICPM)*, pages 9–16. IEEE, 2024.

[36] Adrian Rebmann and Han van der Aa. Enabling semantics-aware process mining through the automatic annotation of event logs. *Information Systems*, 110:102111, 2022.

[37] Diana Sola, Christian Warmuth, Bernhard Schäfer, Peyman Badakhshan, Jana-Rebecca Rehse, and Timotheus Kampik. Sap signavio academic models: a large process model dataset. In *International Conference on Process Mining*, pages 453–465. Springer, 2022.

[38] Han van der Aa, Claudio Di Ciccio, Henrik Leopold, and Hajo A Reijers. Extracting declarative process models from natural language. In *Advanced Information Systems Engineering: 31st International Conference, CAiSE 2019, Rome, Italy, June 3–7, 2019, Proceedings 31*, pages 365–382. Springer, 2019.

[39] Han van der Aa, Adrian Rebmann, and Henrik Leopold. Natural language-based detection of semantic execution anomalies in event logs. *Information Systems*, 102:101824, 2021.

[40] Wil MP van der Aalst. Foundations of process discovery. In *Process Mining Handbook*, pages 37–75. Springer, 2022.

[41] Wil MP Van der Aalst and Ana Karla A de Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, 2005.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].

[43] David Vaughn. To mask or not to mask: The effect of prompt tokens on instruction tuning. `https://towardsdatascience.com/to-mask-or-not-to-mask-the-effect-of-prompt-tokens-on-instruction-tuning-016f85fd67f4/`, 2024. Accessed: 2025-04-26.

[44] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*, 2022.

[45] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[47] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.

[48] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, March 2025. arXiv:2303.18223 [cs].

[49] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pages 12697–12706. PMLR, 2021.