LLMs that Understand Processes: Instruction-tuning for Semantics-Aware Process Mining

Vira Pyrih
Faculty of Computer Science
University of Vienna
Vienna, Austria
a12228590@unet.univie.ac.at

Adrian Rebmann
SAP Signavio
Berlin, Germany
adrian.rebmann@sap.com

Han van der Aa
Faculty of Computer Science
University of Vienna
Vienna, Austria
han.van.der.aa@univie.ac.at

***Abstract*—Process mining is increasingly using textual information associated with events to tackle tasks such as anomaly detection and process discovery. Such semantics-aware process mining focuses on what behavior should be possible in a process (i.e., expectations), thus providing an important complement to traditional, frequency-based techniques that focus on recorded behavior (i.e., reality). Large Language Models (LLMs) provide a powerful means for tackling semantics-aware tasks. However, the best performance is so far achieved through task-specific fine-tuning, which is computationally intensive and results in models that can only handle one specific task. To overcome this lack of generalization, we use this paper to investigate the potential of instruction-tuning for semantics-aware process mining. The idea of instruction-tuning here is to expose an LLM to prompt-answer pairs for different tasks, e.g., anomaly detection and next-activity prediction, making it more familiar with process mining, thus allowing it to also perform better at unseen tasks, such as process discovery. Our findings demonstrate a varied impact of instruction-tuning: while performance considerably improved on process discovery and prediction tasks, it varies across models on anomaly detection tasks, highlighting that the selection of tasks for instruction-tuning is critical to achieving desired outcomes.**

***Index Terms*—semantics-aware, large language models, anomaly detection, next-activity prediction, process discovery**

I. Introduction

Process mining leverages event data to analyze and improve the execution of business processes, supporting tasks such as process discovery, anomaly detection, and next-activity prediction. Traditionally, process mining techniques have relied heavily on frequency-based methods, focusing on how often certain sequences of activities occur in event logs. Recently, a growing body of research has begun to explore semantics-aware process mining, which incorporates the meaning of activities—often expressed through textual labels—into analysis [1], [2], [3]. This shift is driven by advances in natural language processing (NLP), particularly the emergence of large language models (LLMs) with strong capabilities in understanding and generating human language.

Despite the promise of semantics-aware process mining and initial work using fine-tuned LLMs for individual tasks [2], [4], [5], such as anomaly detection and process discovery, current approaches remain limited by their lack of generalization. Fine-tuning requires separate models for each task and format, leading to inflexible solutions that cannot be used across tasks.

These issues can be overcome through *instruction-tuning*, which is a method for fine-tuning LLMs using exemplary pairs of prompts and desired outputs [6], covering different NLP tasks. This enables LLMs to generalize across tasks, by aligning their internal representations with the meaning of the instructions, rather than solely relying on patterns seen in task-specific fine-tuning. As a result, instruction-tuned models can adapt to tasks and prompts not seen during training. Although instruction-tuning has shown success in other domains [7], its effectiveness has not yet been explored in process mining.

Therefore, we present the first systematic investigation of instruction-tuned LLMs for semantics-aware process mining, aiming to answer the question: *Can instruction-tuning be used to improve the performance of LLMs on unseen process mining tasks?* Essentially, we thus assess whether exposing an LLM to certain process mining tasks, such as anomaly detection and next-activity prediction, helps it learn underlying behavioral relations among activities, which can then be leveraged for other tasks, like process discovery. To operationalize this, we build on previous work that introduced benchmark tasks and datasets for evaluating LLMs in this domain [5]. In this manner, we evaluate the ability of instruction-tuned LLMs to generalize across various, control-flow-oriented process mining tasks, including classification tasks such as anomaly detection and next-activity prediction, and generative tasks in the form of process discovery (for both DFGs and process trees). Our experiments compare models that have been instruction-tuned to both untuned and task-specific models.

Our findings suggest that instruction-tuning is a promising path toward more scalable applications of LLMs for semantics-aware process mining, particularly for discovery and predictive tasks. For these tasks, instruction-tuned models exhibit improved generalization capabilities, but our results also reveal challenges in generalization for anomaly detection. The main contribution of this work is the first empirical evidence that a single, instruction-tuned LLM can generalize across multiple semantics-aware process mining tasks, challenging the paradigm of single-task fine-tuning. This has the practical implication of enabling more flexible and scalable process analysis tools, while also defining the research challenge of improving generalization for classification-oriented tasks.

The remainder of this paper is organized as follows: Section II introduces key concepts and Section III outlines the targeted process mining tasks. Section IV details our instruction-tuning approach for LLMs. Section V and Section VI describe the experimental setup and results, respectively. We discuss related work in Section VII and conclude in Section VIII.

II. Preliminaries

This section introduces preliminaries used in the remainder.

Event Data. We adopt a simple event model that focuses on the control-flow of a process. A trace $\sigma = \langle a_1, \ldots, a_n \rangle$ captures a single process instance as a sequence of activities $a_i \in \mathcal{A}$, where $\mathcal{A}$ is the universe of possible activities. 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.

Directly-Follows Graphs. A directly-follows graph (DFG) captures which activities in a process can (or have been observed to) directly succeed each other. We define a DFG as a tuple $D = (A, F)$, with A as the DFG's nodes and F as a set of ordered pairs corresponding to its edges. Each edge $(x, y) \in F$ indicates that an activity $x \in A$ can be directly followed by activity $y \in A$, which we also denote as $x > y$.

Process Trees. A process tree is a hierarchical representation of a process, using activities and a set of operators $O = \rightarrow, \times, \wedge, \circlearrowleft$, denoting sequence, choice, parallelism, and looping. Leaves are activities (or silent activity τ); internal nodes define behavior recursively [8].

Eventually-Follows Relations. An eventually-follows relation is a more relaxed ordering relation than the directly-follows relation used in DFGs, focusing on activities that can either directly or indirectly follow each other. For a trace $\sigma = \langle a_1, ..., a_n \rangle$, any activity pair a_i, a_j with $1 \leq i < j \leq n$ is said to be in an eventually-follows relation, denoted as $a_i \prec a_j$.

III. Semantics-aware Process Mining Tasks

Our work builds on five semantics-aware mining tasks introduced in earlier work [5]. These are designed to assess the ability of language models to reason about control-flow behavior based solely on the semantics of activities, without access to historical event data. They are defined as follows:

Trace-Level Semantic Anomaly Detection (T-SAD). Semantic anomaly detection involves assessing whether observed process behavior makes logical sense or not. In this regard, T-SAD is a classification task where a trace σ, provided along with a set of possible process activities $A \subseteq \mathcal{A}$, must be classified as being either semantically correct or anomalous. For instance, trace $\sigma = \langle$*register application*, *approve application*, *review application*$\rangle$ should be classified as anomalous, since an application should be reviewed before it is approved. Note that including the set A as input enables the identification of anomalies involving missing activities.

Activity-Level Semantic Anomaly Detection (A-SAD). A-SAD is a more fine-granular task, where, given a trace $\sigma = \langle a_1, \ldots, a_n \rangle$ and a set of possible activities A, each eventually-follows relation $a_i \succ a_j$, with $1 \leq i < j \leq n$, should be classified as semantically correct or anomalous. For instance, for the aforementioned trace, the relation *register application* $\succ$ *approve application* should be recognized as valid and *approve application* $\succ$ *review application* as anomalous.

Semantic Next-Activity Prediction (S-NAP). S-NAP is a classification task in which, given a incomplete trace $\sigma = \langle a_1, \ldots, a_n \rangle$, the task is to select the most suitable activity a_{n+1} from a set of possible activities A. For instance, given $\sigma = \langle$*create PO*, *approve PO*$\rangle$ and $A = \{$*create PO*, *approve PO*, *create invoice*, *make payment*$\}$, the next activity should be *create invoice*.

Semantic Directly-Follows Graph Discovery (S-DFD). S-DFD is a generation task[1] in which, given a set of activities A, the goal is to produce a graph $D = (A, F)$, where F contains all valid directly-follows relations between activities in A. For instance, given a set of activities $\{$*create PO*, *approve PO*, *reject PO*, *create invoice*$\}$, the semantic DFG should include edges such as (*create PO*, *approve PO*) and (*create PO*, *reject PO*), whereas it should not include (*reject PO*, *create invoice*).

Semantic Process Tree Discovery (S-PTD). S-PTD is a more complex discovery task than S-DFD. Specifically, given a set of activities A, the goal is to generate a process tree whose structure reflects the behavioral constraints in a process, such as parallelism, choices, and sequential behavior. For instance, given the set $\{$*create PO*, *approve PO*, *reject PO*, *create invoice*$\}$, the generated tree should be equivalent to $\rightarrow$(create PO, $\times$(reject PO, $\rightarrow$(approve PO, create invoice))), capturing that, after creating a PO, it can be either rejected or approved, where the latter case leads to an invoice being created.

IV. Instruction-tuning for Semantics-Aware Process Mining

In this section, we explain how to specialize LLMs for semantics-aware process mining tasks using instruction-tuning, also contrasting it to the in-context learning and fine-tuning methods that have been used in previous works. We begin by outlining these different specialization strategies (Section IV-A), before describing the development of our process mining instruction dataset (Section IV-B).

A. Specializing LLMs for Semantics-Aware Process Mining

Neural language models based on the Transformer architecture [9], [10] generally fall into two categories. Bidirectional models, often called encoders, are typically pretrained using masked language modeling, where they predict masked tokens using context from both directions. Unidirectional models, known as decoders, are trained with autoregressive language modeling, predicting the next token based solely on the preceding ones. LLMs represent large instances of these decoder models, with billions of parameters. Their initial development involves large-scale autoregressive pretraining, which equips them with broad natural language understanding capabilities. After initial pretraining, LLMs can be specialized for specific tasks. Three common strategies for this are illustrated in Figure 1 and described next.

[1]This is a generation task because an answer must be generated, rather than selected from a set of options, as done for the classification tasks.

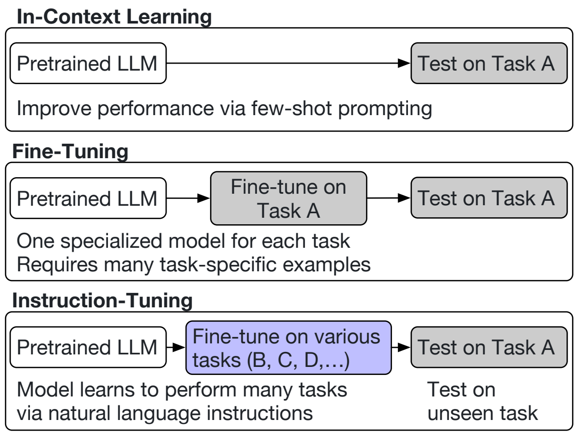

In-Context Learning
Pretrained LLM
Test on Task A
Improve performance via few-shot prompting
Fine-Tuning
Pretrained LLM
Fine-tune on Task A
Test on Task A
One specialized model for each task
Requires many task-specific examples
Instruction-Tuning
Pretrained LLM
Fine-tune on various tasks (B, C, D,...)
Test on Task A
Model learns to perform many tasks via natural language instructions
Test on unseen task

Fig. 1. Comparison between in-context learning, fine-tuning, and instruction-tuning. Adapted from Wei et al. [6]

In-Context Learning. ICL offers a mechanism to elicit task-specific capabilities from an LLM without modifying its underlying parameters. This is done using a prompt that typically includes a natural language description of the task, followed by a few illustrative examples ("shots"), consisting of input-output pairs that demonstrate the desired solution. The actual query instance for which a solution is needed is appended to this context, and the LLM generates the output autoregressively, leveraging the provided examples to infer the task pattern [11]. ICL relies on a model's pretrained knowledge to perform tasks. The model's reasoning is prompted based on a few provided examples. For instance, for the A-SAD task, an LLM is asked to classify a set of eventually-follows pairs as anomalous or not, while also providing it with a few examples of valid and invalid pairs.

Fine-Tuning. Fine-tuning adapts a pretrained language model to a specific task by continuing its training on a labeled dataset relevant to that task, updating the model's parameters accordingly. This is done by providing the model with an extensive set of labeled task instances, such as eventually-follows pairs of activities along with the correct label (*Valid* or *Anomalous*) for A-SAD, yet typically without providing task instructions. In this manner, fine-tuning allows models to achieve considerably higher performance on the targeted task compared to ICL [5]. However, a clear downside of this strategy is that obtained models can only be used for the specific task they were fine-tuned on.

Instruction-Tuning. Instruction-tuning enables LLMs to follow natural language instructions and provide solutions to described tasks. Typically, LLMs are already instruction-tuned following their initial autoregressive pretraining, using datasets containing a wide array of common NLP tasks that are described via natural language instructions [7] . This step aims to improve generalization across diverse capabilities, covering tasks such as text classification, summarization, question answering, information extraction, and text rewriting [12].

While such general instruction-tuning provides broad instruction-following competence, domain-specific instruction-tuning represents a more focused specialization strategy. This involves further training the model on a curated dataset comprising instructions for solving tasks that are relevant to the domain, which, in our case, is process analysis. Domain-specific tasks thus include, e.g., "Classify this trace as anomalous or valid" and "Given this partial trace and list of possible activities, predict the next one", paired with corresponding domain-specific inputs such as traces or activity lists and the desired outputs. Unlike single-task fine-tuning, which primarily optimizes performance for a fixed input-output format, domain-specific instruction-tuning focuses on enhancing the model's ability to interpret the intent behind various instructions phrased using process mining terminology and concepts. The goal is to improve the LLM's zero-shot or few-shot generalization capabilities across different types of process analysis tasks, making it more versatile compared to models relying solely on generic instruction following learned during initial training or adapted via ICL.

B. *Creating a Process Mining Instruction Dataset*

A prerequisite for instruction-tuning is a dataset of labeled instruction-task instances. This section describes how we established such a dataset for semantics-aware process mining tasks, which is publicly available [13].

Labeled-instance Datasets. As a basis for generating our instruction dataset, we use five datasets from our earlier work [5], each consisting of labeled instances of one of the semantics-aware process mining tasks (Section III). They were derived from a collection of over 15,000 process models from the SAP-SAM collection [14], covering a wide range of process types and domains.[2]

To avoid the inclusion of duplicate task instances, we cleaned these available datasets using task-specific rules:

- T-SAD: Filtered for unique (activity, full trace) combinations (184,004 instances).
- A-SAD: Filtered for unique (activity, activity pair) combinations (316,308 instances).
- S-NAP: Removed completed traces and duplicate prefixes for the same next activity, preserving cases with multiple valid outcomes (575,339 instances).
- S-DFD/S-PTD: Filtered for unique input activity sets (15,580 instances).

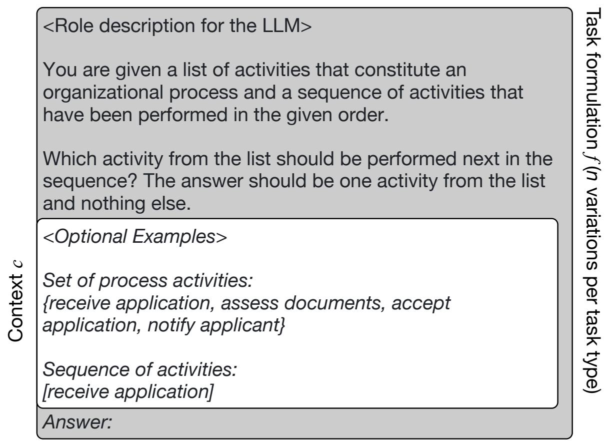
Creating Instruction-Task Instances. Since the labeled-instance datasets only include input and output pairs (t, o) (e.g., for T-SAD, t is a trace with its corresponding output label o as *Valid* or *Anomalous)*, rather than prompts, we create instruction-task instances by enriching the available instances with natural language instructions to be used by an LLM.

To this end, we follow the *human-oriented instructions* approach [15], which aims to establish instructions that are understandable by non-expert users and typically take the form of descriptive, paragraph-style text. They include a brief statement that frames the identity of the model, contextual information such as task descriptions, definitions, and specific

[2]For details on their establishment and characteristics, we refer to [5].

instructions, which has been shown to enhance generalization particularly well [15].

In line with the state of the art [12], [16], each labeled instruction-task instance is a pair $\phi = (i, o)$ consisting of an instruction i, which we detail below, and an expected output o, which we directly derive from the output label o of the original task instance (reformatted as a natural language response). As illustrated in Figure 2 for S-NAP, an instruction-task instance is a tuple $i = (f, c)$. The task formulation f is a natural language description of the task, including the role description of the LLM, requirements for the output format, and a closing phrase prompting the model to generate the desired result. The context c is a textual representation of the task instance (e.g., a trace or activity set) and any additional contextual information (e.g., examples) that the model must process.


<Role description for the LLM>

You are given a list of activities that constitute an organizational process and a sequence of activities that have been performed in the given order.

Which activity from the list should be performed next in the sequence? The answer should be one activity from the list and nothing else.

<Optional Examples>

Set of process activities:
{receive application, assess documents, accept application, notify applicant}

Sequence of activities:
[receive application]

Answer:

Fig. 2. Instruction-task-instance for the S-NAP task.

To increase the diversity and robustness of the resulting instruction dataset, we adhere to the following best practices [7]:
Varying task formulations. We establish six formulations for each task. This exposes the model to different phrasings of the same underlying problem, thereby improving its generalization capabilities to new, unseen instructions. For instance, task formulations for A-SAD range from a formal instruction to “Determine whether it is valid for the first activity to occur before the second”, which presents the activities separately and constrains the output to “Provide either True or False as the answer and nothing else” to more direct phrasings that embed the activities within the prompt, such as “Is the order (first: act1, second: act2) acceptable...?” while requesting the model to simply “Answer with True or False.” The complete set of variations for all tasks is available in our repository.
Incorporating instructions about negative or undesired behavior. Real-world process mining often involves identifying or reasoning about undesired, incorrect, or incomplete process behavior. To reflect this, we create instructions that focus on such negative aspects where appropriate. Here, *negative* refers to instructions prompting the model to identify, complete, or reason about incorrect, incomplete, or undesirable process elements or sequences. For example, for S-NAP, a negative task instance might describe an ongoing process execution that is missing an important activity that should have occurred earlier. The model is then tasked to identify this missing activity from any point in the prior sequence leading up to the last observed activity. This helps prevent LLMs from overfitting solely to desired process behavior and ensures they can handle the variance seen in practical applications.
Inverting task objectives. We generate reversed versions of a given task by flipping its objective, if the task structure allows. This can lead to both *positive* and *negative* inversions.

- *Positive inversion* typically involves the model generating or identifying correct, preceding, or completing elements for an assumed valid partial context. For example, for S-NAP, a positive inversion involves providing the model with a known subsequent activity in a trace and asking it to generate a plausible sequence of activities that might have occurred *before* this given activity. For A-SAD, a positive inversion might involve completing a valid partial activity pair by having the model choose an activity that can legitimately follow a given one.
- *Negative inversion* involves the model identifying elements that render a task instance incorrect, highlight what should not occur, or complete a scenario in a way that demonstrates an invalid or undesirable path. For instance, a negative inversion of A-SAD, could involve asking the model to choose an activity that would create an invalid eventually-follows pair if it followed a given activity.

For T-SAD and A-SAD, the nature of the inversion depends on whether the original task instance presented is valid or anomalous. For instance, if an A-SAD instance is an anomalous pair, a negative inversion asks to create an activity pair such that it is anomalous, while a positive inversion for a valid pair asks to create a valid pair. For S-NAP, the inversion is chosen to be positive or negative with equal probability. For the discovery tasks, since generating a positively inverted task is not meaningful (e.g., generating a set of activities from a given DFG), we only apply negative inversions.

Following these best practices, we transform each original task instance into a labeled instruction-task instance ϕ. We begin by randomly selecting a task formulation f from the available ones. The original task instance t is then transformed into a corresponding context description c (occassionally using inversion). These components, f and c, form the instruction pair $i = (f, c)$, which, together with the original output label o, constitutes the labeled instruction-task instance $\phi = (i, o)$. In this way, we obtain an instruction data set, made publicly available [13], whose characteristics are shown in Table I.

TABLE I
CHARACTERISTICS OF THE INSTRUCTION DATASET.

Task	Samples (Total)	Normal (%)	Neg. Inv. (%)	Pos. Inv. (%)
A-SAD	316,308	80	10	10
T-SAD	184,004	80	10	10
S-NAP	575,339	80	10	10
S-DFD	15,580	80	20	0
S-PTD	15,580	100	0	0

V. Experimental Setup

We evaluate the generalization capabilities of LLMs in solving semantics-aware process mining tasks, primarily examining the difference between models with and without domain-specific instruction-tuning. To this end, this section details our experimental setup; the code used to conduct the experiments is available in our repository.[3]

Task Grouping. We evaluate the LLMs' ability to generalize across different process mining objectives by grouping the tasks into three coherent groups: *Anomaly* (A-SAD and T-SAD), *Prediction* (S-NAP), and *Discovery* (S-DFD and S-PTD). This enables a leave-one-group-out evaluation setup, where models are trained on two groups (e.g., Anomaly and Prediction) and tested on the held-out group (e.g., Discovery). By using groups rather than individual tasks, we avoid bias caused by having an anomaly detection (or discovery) task in both the training and the test set.

Dataset Splitting. To ensure reproducibility and consistent comparisons, we construct our leave-one-group-out folds using fixed data splits for each task, adopted from our previous work [5]. For each task, instruction instances are derived from a distinct set of underlying process models. These models are partitioned to allocate 70% for generating training instances, 20% for validation, and 10% for testing. These pre-defined, task-specific splits are then combined to form the datasets for each leave-one-group-out fold as follows:

Training. For a given fold, the training data is formed by combining the training splits of the tasks belonging to the two in-fold groups.

Validation. The combined validation splits of tasks in the held-out group guide model checkpoint selection during training.

Test. The combined test splits of tasks in the held-out group guide the evaluation of final model performance.

Building on fixed individual task splits guarantees that, for instance, the test set for the *Discovery* group remains identical, whether used in our leave-one-group-out evaluation or by other researchers for direct comparisons on S-DFD and S-PTD tasks.

Training Data Sampling. To balance task representation for instruction-tuning, we sample training sets using examples-proportional mixing [17] from constituent tasks, generally capping contributions at 30,000 samples per task. This cap is adjusted when holding out the *Discovery* group: S-NAP contributes 60,000 samples to balance the 60,000 from two anomaly detection tasks. The characteristics of the resulting training folds are shown in Table II.

Large Language Models. For our instruction-tuning experiments, we selected two powerful decoder language models: Llama 3 70B Instruct[4] and Mistral Large 2 Instruct (v. 2407)[5]. These were chosen for their robust instruction-following capabilities, model size, and open-source availability.

Instruction-Tuning Procedure. We use parameter-efficient fine-tuning based on quantized low-rank adaptation (LoRA) [18], using pre-quantized 4-bit models, configuring LoRA with rank $r = 16$ and scaling factor $\alpha = 16$ to balance model expressiveness while minimizing the risk of overfitting. We verified that higher ranks did not lead to consistent improvements. We set the learning rate to 1×10^{-5} and adjust it dynamically using a linear scheduler. We perform training with a batch size of 8 per GPU and 4 gradient accumulation steps, resulting in an effective batch size of 32[6]. We fine-tune for 3 epochs, using a maximum sequence length of 1024 tokens. We use the AdamW optimizer with a weight decay of 0.01 to improve regularization.

During training, we use standard cross-entropy as the loss-function and monitor validation performance every 250 steps using the combined validation splits of tasks in the held-out group. For model selection, instead of relying solely on cross-entropy loss, we compute task-specific custom metrics that are better aligned with the target tasks. When validating multiple tasks simultaneously (for the Anomaly and Discovery groups), we average the metric improvements across tasks and select the checkpoint achieving the highest overall improvement.

Performance Measures. We assess the obtained results using established performance measures:

Anomaly Detection and Prediction Performance. Since anomaly detection (T-SAD and A-SAD) and next-activity prediction (S-NAP) involve classification, we assess them using the macro F_1-score. This measure ensures that all classes of the respective task contribute equally to the overall performance, irrespective of their size. The macro F_1-score is the simple average of the F_1-scores for each individual class (e.g., valid and anomalous traces in T-SAD), where the F_1-score for a class is the harmonic mean of its precision and recall.

Discovery Performance. For the discovery tasks (S-DFD and S-PTD), we measure performance through footprint-based fitness [19]. This measure facilitates the comparison of sets of allowed execution sequences by using pairwise behavioral relations. As such, it considers whether a discovered model allows for the same behavior as the ground-truth model.

TABLE II
Training sample distribution per test group

Test Group	**Training Task**	**Samples**	**Share**	**Prompt Types (%)** Norm.	Neg.Inv.	Pos.Inv.
Anomaly	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**	**14.90**	**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**

[3]https://github.com/pirogtm7/it4pm

[4]https://huggingface.co/unsloth/Llama-3.3-70B-Instruct-bnb-4bit

[5]https://huggingface.co/unsloth/Mistral-Large-Instruct-2407-bnb-4bit

[6]For Mistral's prediction and anomaly tasks, the batch size was reduced to 4 (effective size 16) due to memory constraints.

VI. Experimental Results

We report on the LLMs' performance on the five tasks, comparing their performance with and without instruction-tuning, followed by an in-depth analysis of the results.

A. Main Results

Table III reports on the overall results.[7] We observe mixed impacts of instruction-tuning on performance. While the instruction-tuned models consistently outperform the base models for the prediction (S-NAP) and discovery (S-DFD and S-PTD) tasks, the effect on anomaly detection tasks (A-SAD and T-SAD) differs between models.

TABLE III
Performance across models and tasks.

Task (Metric)	**Llama Base**	**Llama IT**	**Mistral Base**	**Mistral IT**
A-SAD (macro F_1)	0.594	0.562	0.421	**0.679**
T-SAD (macro F_1)	0.558	0.480	0.347	**0.620**
S-NAP (macro F_1)	0.525	**0.651**	0.624	**0.868**
S-DFD (Fitness)	0.630	**0.714**	0.658	**0.770**
S-PTD (Fitness)	0.621	**0.697**	0.649	**0.763**

Specifically, on anomaly detection, Llama IT's performance decreases (T-SAD F_1: 0.558 to 0.480), whereas Mistral IT's improves substantially (T-SAD F_1: 0.347 to 0.620). For example, given the sequence of activities ⟨*Approve application*, *Apply for trip*, *Buy transport tickets*, *Book accommodation*, *Archive trip documents*⟩, the Llama Base correctly identifies the trace as invalid, whereas the Llama IT model incorrectly flags it as valid. This suggests that instruction-tuning on discovery and predictive tasks may have overwritten some of Llama's semantic anomaly capabilities. However, this effect is not universal, as Mistral IT consistently improves performance compared to its base version. Conversely, for S-NAP, instruction-tuning boosts performance for both models. Llama IT's F_1 score increases from 0.525 to 0.651. For instance, given the prefix ⟨*Check stock availability*, *Retrieve product*⟩, Llama Base suggests *Emit invoice*, whereas Llama IT, more logically, predicts *Confirm order*, i.e., that an order must be confirmed before an invoice is issued.

Moving to process discovery, instruction-tuning shows substantial benefits for both tasks. For Llama, the fitness score improves from 0.630 (Base) to 0.714 (IT) for S-DFD. Mistral exhibits even larger gains, with fitness increasing from 0.658 (Base) to 0.770 (IT). Similarly, for the S-PTD task, instruction-tuning leads to considerable improvements. Llama's fitness score rises from 0.621 to 0.697 after instruction-tuning. Mistral sees its fitness score increase from 0.649 to 0.763.

Overall, the results reveal a varied impact of instruction-tuning. While Llama IT shows decreased macro F_1 scores on A-SAD and T-SAD, Mistral IT shows a considerable improvement. Both models, however, achieve notable improvements on S-NAP. For tasks requiring structured output generation, namely S-DFD and S-PTD, instruction-tuning consistently yields substantial gains in fitness scores for both Llama and Mistral. These improvements confirm that both models considerably benefit from domain-specific instruction-tuning in these scenarios. The enhancement suggests that instruction-tuning not only facilitates task-specific adaptation for structured output but also helps models grasp higher-level process representations, particularly where multiple plausible outputs exist. Notably, Mistral IT consistently outperforms Llama IT on these tasks, as expected from its larger size. For instance, Llama IT incorrectly placed *Analyze claim* in sequence after $\times$(*Handle easy claim*, *Handle complex claim*), whereas Mistral IT correctly placed it before, resulting in higher fitness.

B. In-Depth Analysis

Beyond the aggregate performance metrics (Section VI-A), we report on qualitative results of instruction-tuned models and their base counterparts, providing more detailed insights into their understanding of process behavior.

Improved Instruction Adherence. We find that instruction-tuned models consistently show improved adherence to specified instruction formats and produce cleaner, more direct outputs compared to base models. For instance, in the T-SAD task, Llama IT provided clear *True* or *False* responses, whereas Llama Base often returned ambiguous or irrelevant text. Similarly, for S-NAP, Llama IT generated concise next-activity predictions, a considerable improvement over the base model's tendency to generate boilerplate text or code-like structures. This enhanced clarity also extends to the discovery tasks: the instruction-tuned models produced better-formatted directly-follows pairs (S-DFD) and more structurally coherent process trees (S-PTD), largely avoiding malformed or incomplete outputs that are common with base models. This suggests that instruction-tuning effectively guides the models' focus on the core process analysis task and desired output structure.

Superiority in Semantic Discovery Tasks. The benefits of instruction-tuning are particularly visible in semantic discovery tasks (S-DFD and S-PTD), which require a more holistic understanding of the underlying process and activity relationships. While the base models capture basic relations, instruction-tuned models demonstrate a more developed grasp of higher-level process representations. For example, in S-PTD, Mistral IT not only achieved higher fitness but also exhibited a qualitatively better interpretation of control-flow operators such as exclusion ($\times$) and sequence ($\to$) compared to its base version, leading to more semantically sound process trees. This suggests that exposure to diverse process-related instructions helps instruction-tuned models develop a better internal representation of process structures.

Challenges in Classification-Oriented Tasks. For classification tasks such as anomaly detection (A-SAD and T-SAD), the impact of instruction-tuning was model-dependent. For

[7]These results should be considered in light of Mistral's considerably longer tuning times due to its larger architecture (123B vs. 70B parameters). For instance, an epoch for the *Discovery* test group took 37h vs 21h for Llama.

Llama IT, the impact was unfavorable, with the model underperforming its base counterpart. In contrast, Mistral IT showed considerable performance gains on these same tasks. A reason for Llama's less favorable performance lies in a prediction bias for A-SAD and T-SAD towards predicting *True* (valid) for the instruction-tuned model, which Mistral could compensate, likely due to its larger size. For T-SAD, Llama's bias became particularly clear, with 89.1% of predictions being *True*, while the test data has an even label distribution. This indicates that while the IT model improves instruction adherence, it does not resolve issues such as label bias if not carefully managed in the instruction dataset design and task formulation. The difficulty of these tasks, requiring an inference based solely on activity semantics without context information, likely contributes to these challenges. Investigating how to address them remains an important direction for future research.

Domain-Specific Differences. To assess model performance across process domains, we categorized the instruction source process models by industry.[8] Both base and instruction-tuned models exhibit varying performance across different process industries. Instruction-tuned models generally performed well in domains such as *Logistics* and *Education*, which are often characterized by more structured, procedural, and clearly defined processes (e.g., shipment handling, course registration) aligning well with patterns that are learnable by LLMs. Conversely, domains such as *Healthcare* and *IT/Software* present greater challenges. These areas are characterized by flexible workflows, high variability across cases (e.g., patient treatment, software development), and frequent ad-hoc decisions, making it more difficult for LLMs to generalize behavioral patterns. This underscores that while instruction-tuning enhances adaptability, inherent domain complexity and variability remain substantial challenges.

In summary, instruction-tuning is a promising path to more flexible semantics-aware process mining. It particularly enhances performance on semantic discovery and instruction adherence. Challenges persist in classification and highly variable domains, indicating a need for task refinement and additional model context.

C. Comparison to Fine-Tuning

To put the results of our instruction-tuning experiments into context, we compare them with the task-specific fine-tuned (FT) smaller language models from our previous work [5], i.e., the small encoder model RoBERTa[9] and the Mistral 7B and Llama 8B decoder LLMs.

Comparing IT results (Table III) with FT results (Table IV) shows FT models achieve consistently higher scores. For instance, the FT Llama 8B scores 0.88 on the A-SAD task, far surpassing the IT Llama 70B's 0.562. This gap persists even for the much smaller RoBERTa model (scoring 0.85). The smaller fine-tuned LLMs also outperform their 70B+ IT counterparts on both discovery tasks. These results are unsurprising, as FT models are trained on large amounts of in-task data, whereas the IT models have observed none.

TABLE IV
PERFORMANCE OF FINE-TUNED (FT) LLMS.

Task (Metric)	RoBERTa	Mistral 7B	Llama 8B
T-SAD (macro F_1)	0.77	0.79	0.79
A-SAD (macro F_1)	0.85	0.88	0.88
S-NAP (macro F_1)	0.63	0.68	0.69
S-DFD (Fitness)	-	0.81	0.80
S-PTD (Fitness)	-	0.84	0.83

However, this superior performance has a substantial drawback: each FT model is a non-generalizing specialist. This requires creating extensive labeled datasets for every task and introduces considerable operational challenges in deploying, maintaining, and updating these models. Furthermore, for classification tasks, the performance gains of fine-tuning large decoder models over encoders are often marginal. As shown in Table IV, the RoBERTa encoder achieves scores that the FT LLMs only slightly surpass, suggesting a smaller encoder may be a more efficient choice for these tasks.

In contrast, the key advantage of IT models is their ability to generalize to unseen tasks, the capability evaluated in our leave-one-group-out setup. Moreover, the crucial advantage of the LLM (decoder) architecture is its suitability for instruction-tuning, as encoders cannot be instruction-tuned to handle diverse, unseen tasks in the same way.

Overall, the choice between FT and IT thus depends on the desired outcome: maximizing performance on a single, static task, or building a more versatile process analysis tool.

VII. RELATED WORK

The advent of LLMs has resulted in a range of works that explore their potential in the broader process analysis domain, including various applications in process mining [20].

Our work focuses on process mining tasks that can be evaluated using gold-standard benchmarks, enabling rigorous methodological evaluation and fine-tuning. Existing work in this category has predominantly addressed semantic anomaly detection [2], [21], next activity prediction [22], [23] and the discovery of declarative process constraints [4], thus aligning with our A-SAD, T-SAD, S-NAP, and S-PTD tasks (though, the latter only to some extent).

In contrast, a range of studies apply LLMs to tasks lacking gold-standard assessments. While this makes it harder to objectively evaluate model performance, it allows exploration of a broader set of process mining applications. Such works include those focused on identifying bottlenecks or undesired process behaviors [24], abstracting fine-granular events into higher-level ones [25], incorporating domain knowledge into process discovery [3], and providing explanations in prescriptive process monitoring [26]. To address the lack of standardized evaluation in such settings, Berti et al. [27] proposed a benchmark of process mining analysis questions that enables

[8]The detailed categorization procedure and quantitative results are available in our repository.

[9]Note that encoder models do not have text completion capabilities, thus RoBERTa cannot be used for the discovery tasks.

self-evaluation of LLM performance. While promising, the use of self-evaluation has been shown to introduce potential biases [28], thus requiring careful interpretation of results.

VIII. Conclusion

Summary. This work presented the first experimental study evaluating the potential of instruction-tuning to enhance the process mining capabilities of LLMs, aiming to overcome the downsides of task-specific fine-tuning. Our results demonstrate that instruction-tuned models outperform their untuned counterparts on several key process mining tasks. However, the observed performance varies across models in anomaly detection highlights the critical need for careful task selection during the instruction-tuning phase. Notably, the most substantial performance gains were observed in process discovery tasks, which demand a deep understanding of behavioral dependencies within processes. This suggests that exposing LLMs to a diverse set of process mining tasks can significantly improve their process comprehension.

Limitations. The main limitation of our study stems from its controlled experimental design. While necessary for rigorous evaluation, this setup does not fully capture the complexity of real-world settings, as our tasks intentionally focused on the control-flow perspective using structured inputs. Consequently, the robustness of our instruction-tuned models remains untested on noisy, real-world event logs that contain incomplete data or require other perspectives like time and resources. Furthermore, our results for instruction-tuning itself reveal a critical challenge: while it is beneficial for generative tasks, it can hinder performance on anomaly detection. This suggests that the selection of instruction-tuning tasks for classification requires further investigation.

Outlook. Future research should further investigate the applicability of instruction-tuned LLMs to other areas of process mining. In particular, it would be valuable to examine whether the benefits of instruction-tuning on well-defined, gold-standard tasks transfer to tasks lacking such benchmarks (cf. Section VII). This would involve leveraging structured instruction-tuning to improve performance in more subjective or exploratory settings. Additionally, we plan to explore how instruction-tuned LLMs can be integrated with traditional process mining techniques, such as combining frequency-based process discovery with semantic assessments.

Open science: Our evaluation scripts, instruction-tuned models, and more detailed evaluation results are available through our project repository: https://github.com/pirogtm7/it4pm. Our instruction dataset is published separately [13].

References

[1] H. van der Aa, A. Rebmann, and H. Leopold, "Natural language-based detection of semantic execution anomalies in event logs," *Information Systems*, vol. 102, p. 101824, 2021.

[2] J. Caspary, A. Rebmann, and H. van der Aa, "Does this make sense? machine learning-based detection of semantic anomalies in business processes," in *BPM*. Springer, 2023, pp. 163–179.

[3] A. Norouzifar, H. Kourani, M. Dees, and W. M. van der Aalst, "Bridging domain knowledge and process discovery using large language models," *arXiv preprint arXiv:2408.17316*, 2024.

[4] K. Busch, T. Kampik, and H. Leopold, "xSemAD: Explainable semantic anomaly detection in event logs using sequence-to-sequence models," in *BPM*. Springer, 2024, pp. 309–327.

[5] A. Rebmann, F. D. Schmidt, G. Glavaš, and H. van der Aa, "On the potential of large language models to solve semantics-aware process mining tasks," *Process Science*, 2025.

[6] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *ICLR*, 2021.

[7] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu *et al.*, "Instruction tuning for large language models: A survey," *arXiv:2308.10792*, 2023.

[8] W. M. van der Aalst, "Foundations of process discovery," in *Process Mining Handbook*. Springer, 2022, pp. 37–75.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, vol. 30, 2017.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL*. ACL, 2019, pp. 4171–4186.

[11] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey on in-context learning," *arXiv:2301.00234*, 2022.

[12] Y. Wang, S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Arunkumar, A. Ashok, A. S. Dhanasekaran, A. Naik, D. Stap *et al.*, "Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks," in *EMNLP*, 2022.

[13] V. Pyrih, A. Rebmann, and H. van der Aa, "Instruction datasets for process mining," May 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15498373

[14] D. Sola, C. Warmuth, B. Schäfer, P. Badakhshan, J.-R. Rehse, and T. Kampik, "SAP Signavio academic models: A large process model dataset," in *ICPM Workshops*. Springer, 2023, pp. 453–465.

[15] R. Lou, K. Zhang, and W. Yin, "Large language model instruction following: A survey of progresses and challenges," *Computational Linguistics*, vol. 50, no. 3, pp. 1053–1095, 2024.

[16] S. Longpre, L. Hou, T. Vu, A. Webson, H. W. Chung, Y. Tay, D. Zhou, Q. V. Le, B. Zoph, J. Wei *et al.*, "The flan collection: Designing data and methods for effective instruction tuning," in *ICML*. PMLR, 2023, pp. 22 631–22 648.

[17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[18] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models," in *ICLR*, 2021.

[19] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance checking*, 2018, vol. 56.

[20] B. Estrada-Torres, A. del Río-Ortega, and M. Resinas, "Mapping the landscape: Exploring large language model applications in business process management," in *BPMDS*. Springer, 2024, pp. 22–31.

[21] W. Guan, J. Cao, J. Gao, H. Zhao, and S. Qian, "Dabl: Detecting semantic anomalies in business processes using large language models," in *AAAI*, vol. 39, no. 11, 2025, pp. 11 735–11 744.

[22] V. Pasquadibisceglie, A. Appice, and D. Malerba, "Lupin: A LLM approach for activity suffix prediction in business process event logs," in *ICPM*. IEEE, 2024, pp. 1–8.

[23] A. Oved, S. Shlomov, S. Zeltyn, N. Mashkif, and A. Yaeli, "Snap: semantic stories for next activity prediction," in *AAAI*, vol. 39, no. 28, 2025, pp. 28 871–28 877.

[24] A. Berti, D. Schuster, and W. M. van der Aalst, "Abstractions, scenarios, and prompt definitions for process mining with LLMs: A case study," in *BPM*. Springer, 2023, pp. 427–439.

[25] M. Fani Sani, M. Sroka, and A. Burattin, "LLMs and process mining: Challenges in RPA: Task grouping, labelling and connector recommendation," in *ICPM*. Springer, 2023, pp. 379–391.

[26] K. Kubrak, L. Botchorishvili, F. Milani, A. Nolte, and M. Dumas, "Explanatory capabilities of large language models in prescriptive process monitoring," in *BPM*, vol. 14940. Springer, 2024, pp. 403–420.

[27] A. Berti, H. Kourani, and W. M. van der Aalst, "PM-LLM-Benchmark: Evaluating large language models on process mining tasks," *arXiv:2407.13244*, 2024.

[28] A. Panickssery, S. R. Bowman, and S. Feng, "LLM evaluators recognize and favor their own generations," *preprint arXiv:2404.13076*, 2024.