PM4Py.LLM: a Comprehensive Module for Implementing PM on LLMs

Alessandro Berti[1,2]

[1] Process and Data Science Chair, RWTH Aachen University, Aachen, Germany
[2] Fraunhofer FIT, Sankt Augustin, Germany
alessandro.berti@fit.fraunhofer.de

Abstract. *pm4py* is a process mining library for Python implementing several process mining (PM) artifacts and algorithms. It also offers methods to integrate PM with large language models (LLMs). This paper examines how the current paradigms of PM on LLM are implemented in *pm4py*, identifying challenges such as privacy, hallucinations, and the context window limit.

Keywords: pm4py · Python · Large Language Models · Large Vision Language Models

1 Introduction

Process mining (PM) is a branch of data science aiming to extract process-related insights from the event data recorded by information systems supporting the execution of such processes. Both open-source (ProM) and commercial (Fluxicon Disco, Celonis, SAP Signavio) tools are available for PM. Among open-source offerings, the *pm4py* process mining library for Python [3] is a popular option implementing various process mining techniques, including process discovery, conformance checking, and model enhancement. *pm4py*, being a library, implements several PM artifacts, including event log structures, process models (process trees, Petri nets, BPMN models), and machine learning features. Large Language Models (LLMs) and Large Vision Language Models (LVLMs) have been recently implemented for PM tasks in different implementation paradigms [1]. The goal of this paper is to present how *pm4py*, allowing for the implementation of different paradigms, integrates with large language models. In [2], some textual abstractions of event logs and process models provided by *pm4py* are described. However, many more techniques (visual abstractions, APIs for code generation) are currently provided in the library, that are introduced in this paper and are important to extend the scope of PM on LLMs. The rest of the paper is organized as follows. Section 2 describes the current paradigms for PM on LLMs; Section 3 describes the implementation of the paradigms in *pm4py*; Section 4 concludes the paper with an outlook over possible development directions.

2 Paradigms for PM on LLMs

The following paradigms have been adopted for PM on LLMs [1]:

- *Direct Provision of Insights*: A prompt containing abstractions of PM artifacts, domain knowledge, and questions is provided to the LLM/LVLM. Different techniques are available for the abstraction of PM artifacts:
  - *Textual Abstractions of Process Mining Artifacts*: The PM artifact is provided as text [2].
  - *Visual Abstractions of Process Mining Artifacts*: The PM artifact is provided as a visual representation, which can be interpreted by the LVLM.

  Typical questions that are possible using this paradigm are: "What are the anomalies in the event data?", "Can you explain the process?". The main limitation is the context window limit of the LLM/LVLM. Moreover, the proposed implementation paradigm might suffer from privacy (data needs to be provided to the LLM) and *hallucinations* by the LLM/LVLM [8]. In *pm4py*, the textual/visual abstractions can be provided along with a question to an LLM. In particular, `pm4py.llm.openai_query` is implemented, which allows querying OpenAI's LLMs/LVLMs (such as *gpt-4-turbo-preview* and *gpt-4-vision-preview*).
- *Translation of Natural Language Statements to SQL Queries*: Alternatively, the LLM can generate SQL queries executable on the process mining dataset [4]. This mitigates the privacy and hallucination issues. However, the user needs to provide a clear question and all the required domain knowledge on the dataset and on the process [4]. Typical questions that are possible using this paradigm are: "How many cases are contained in this event log?", "Can you measure the average throughput time of the cases containing the activity X?".
- *Code Generation*: Using the LLM to generate scripts executable against the process mining dataset is a paradigm applied successfully in process modeling [5]. It requires the detailed specification of the APIs to be used. Typical statements that are possible using this paradigm are: "Discover a process model from the data", "Extract an event log from my Outlook calendar".
- *Automatic Formulation of Hypotheses*: This paradigm provides textual abstractions of the process mining artifacts in order for the LLM to generate hypotheses over the artifact [2]. A SQL query is also provided to verify the validity of such hypotheses. Refinement cycles can be realized when the original hypotheses are not valid, leading to refined hypotheses.

The provision of the original process mining artifact is usually not a viable option due to the context window limit, as real-life event logs and process models have millions of events and hundreds of activities.

3 Implementations

In this section, the implementation in *pm4py* of the different paradigms described in Section 2 is discussed.

Command	Description
`pm4py.llm.abstract_dfg`	Provides the DFG abstraction of a traditional event log.
`pm4py.llm.abstract_variants`	Provides the variants abstraction of a traditional event log.
`pm4py.llm.abstract_log_attributes`	Provides the abstraction of the attributes/columns of the event log.
`pm4py.llm.abstract_log_features`	Provides the abstraction of the machine learning features obtained from an event log.
`pm4py.llm.abstract_case`	Provides the abstraction of a case (collection of events).
`pm4py.llm.abstract_ocel`	Provides the abstraction of an object-centric event log (list of events and objects).
`pm4py.llm.abstract_ocel_ocdfg`	Provides the abstraction of an object-centric event log (OC-DFG).
`pm4py.llm.abstract_ocel_features`	Provides the abstraction of an object-centric event log (features for ML).
`pm4py.llm.abstract_event_stream`	Provides an abstraction of the (last) events of the stream related to a traditional event log.
`pm4py.llm.abstract_temporal_profile`	Provides the abstraction of a temporal profile model.
`pm4py.llm.abstract_petri_net`	Provides the abstraction of a Petri net.
`pm4py.llm.abstract_declare`	Provides the abstraction of a DECLARE model.
`pm4py.llm.abstract_log_skeleton`	Provides the abstraction of a log skeleton model.

Table 1: Textual abstractions of PM artifacts provided by *pm4py*.

3.1 Textual Abstractions of Process Mining Artifacts

Textual abstractions of PM artifacts can be provided (up to the context window limit) to any LLM. The possible abstractions are described in Table 1. For traditional event logs, the top entries of the directly-follows graph (couples of activities following each other in at least a case of the log) or the top process variants can be computed. Moreover, it is possible to extract a summary of the numerical machine learning features extracted from the event log (each feature is provided with different quantiles). For object-centric event logs, the top entries of the object-centric directly-follows graph can be extracted. As for traditional event logs, it is possible to extract a summary of the machine learning features extracted from the object-centric event log. It is also possible to abstract the entire list of events and objects of the object-centric event log. However, the resulting textual abstraction would probably exceed the context window limit of the LLM if not limited to a subset of events/objects. The methods support the specification of the context window limit. The abstractions usually provide, at first, the most important entry and continue to include entries until the specified length limit is exceeded.

Petri nets can be textually abstracted as their set of places/transitions/arcs [2]. Declarative process models, such as DECLARE [7] and the log skeleton [9], can also be textually abstracted. The exact specification of the artifact can vary. However, since LLMs might not have the knowledge of a specific declarative notation, the description of the rules of the notation should also be included in the textual abstraction. In contrast to event log abstractions, it is more difficult to enforce the context window limit because of missing criteria to rank the importance of the elements of the process model. Textual abstractions of large Petri nets or declarative models with many rules can be provided only to LLMs with a large context window.

3.2 Visual Abstractions of Process Mining Artifacts

Visual process representations can also be a source of knowledge for PM on LVLMs. The method

```
pm4py.llm.explain_visualization(vis_saver, *args, connector, **kwargs)
```

Command	Description
`pm4py.vis.save_vis_petri_net`	Saves the visualization of a Petri net model.
`pm4py.vis.save_vis_dfg`	Saves the visualization of a directly-follows graph annotated with the frequency.
`pm4py.vis.save_vis_performance_dfg`	Saves the visualization of a directly-follows graph annotated with the performance.
`pm4py.vis.save_vis_process_tree`	Saves the visualization of a process tree.
`pm4py.vis.save_vis_bpmn`	Saves the visualization of a BPMN model.
`pm4py.vis.save_vis_heuristics_net`	Saves the visualization of an heuristics net.
`pm4py.vis.save_vis_dotted_chart`	Saves the visualization of a dotted chart.
`pm4py.vis.save_vis_sna`	Saves the visualization of the results of a social network analysis.
`pm4py.vis.save_vis_case_duration_graph`	Saves the visualization of the case duration graph.
`pm4py.vis.save_vis_events_per_time_graph`	Saves the visualization of the events per time graph.
`pm4py.vis.save_vis_performance_spectrum`	Saves the visualization of the performance spectrum.
`pm4py.vis.save_vis_events_distribution_graph`	Saves the visualization of the events distribution graph.
`pm4py.vis.save_vis_ocdfg`	Saves the visualization of an object-centric directly-follows graph.
`pm4py.vis.save_vis_ocpn`	Saves the visualization of an object-centric Petri net.
`pm4py.vis.save_vis_object_graph`	Saves the visualization of an object-based graph.
`pm4py.vis.save_vis_network_analysis`	Saves the visualization of the results of a network analysis.
`pm4py.vis.save_vis_transition_system`	Saves the visualization of the results of a transition system.
`pm4py.vis.save_vis_prefix_tree`	Saves the visualization of a prefix tree.
`pm4py.vis.save_vis_alignments`	Saves the visualization of the alignments table.
`pm4py.vis.save_vis_footprints`	Saves the visualization of the footprints table.
`pm4py.vis.save_vis_powl`	Saves the visualization of a POWL model.

Table 2: Visualization methods provided by *pm4py*.

can be used to provide a visualization to an LVMs, in which `vis_saver` is one of the visualizations described in Table 2, ∗`args` (positional arguments) and ∗∗`kwargs` (keyword arguments) are the arguments provided to the visualization method, and `connector` is the LVM that should be used. For instance, `pm4py.llm.openai_query` can be used to query the *gpt-4-vision-preview* model. Several visualizations offer insights that are more immediate in comparison to the textual abstractions, for instance:

– Dotted charts show events and detect patterns like batching and concept drifts.
– Performance spectrum visualizes activity flows over time, offering performance insights.
– Case duration and events per time graphs help understand throughput and identify concept drifts.
– Petri nets, BPMN, directly-follows graphs, and social network analysis benefit from visual layout for pattern detection.

It is important to acknowledge that prompts to LVLMs require more computationally intensive operations than textual prompts to LLMs. Moreover, LVLMs are not specialized in patterns that might be important for process mining, as highlighted in [1]. Moreover, LVLMs may miss domain knowledge needed to interpret such visualizations (for instance, additional explanation may be required in the prompt to detect some patterns on the dotted charts).

3.3 SQL Queries against *pm4py* Event Logs

Statements in natural language can be translated by LLMs in the most important SQL dialects [6]. The data structure used for traditional event logs in *pm4py* is the Pandas dataframe, which is a columnar data structure easily imported/-exported from/to CSV and Parquet files. Pandas dataframes can be queried in-memory using, for instance, the DuckDB SQL dialect. The LLM needs to be provided with the attributes of the Pandas dataframe and their specification (case ID, activity, timestamp). Moreover, if the target LLM's knowledge of DuckDB is not proficient, additional knowledge related to the database operators and process mining need to be provided. The SQL query can be executed using `duckdb.sql(sqlquery).to_df()`, which returns a Pandas dataframe containing the answer to the question of the user. It is important to note that only quantitative statements can be translated to SQL queries using this paradigm. Also, the object-centric data structure in *pm4py* is a collection of Pandas dataframes (there are dataframes for the events, objects, and event-to-object relationships). Therefore, SQL queries can be orchestrated against any combination of the dataframes part of the object-centric data structure, enabling LLM-based querying of object-centric event logs[3]

3.4 Generating *pm4py* Code with LLMs

pm4py comes with a rich documentation covering its features[4]. Using *RAG* or *fine-tuning*, an LLM can be trained over the API of the library[5], becoming able to translate user statements in executable Python code. This paradigm allows not only to query the event data but also to apply the algorithms provided by *pm4py* and from any scientific Python library in general. However, it might pose a security risk as code generated by the LLM could contain malicious instructions [10]. An example of LLM trained to generate *pm4py* code is provided in the OpenAI's GPT store[6].

3.5 Automatic Formulation of Hypotheses

Combining the paradigms described in Section 3.1 and 3.3, it is possible to generate some hypotheses over the process mining artifacts, in particular concerning event data. Usually, some information about the activities (for example, the textual abstraction of the directly-follows graph) and the attributes of the event log are provided along with a question to generate hypotheses. The request can generically target any hypothesis over the event data or hypotheses over a specific domain.

An example script is provided for hypothesis generation[7]. Each hypothesis comes with a description/explanation and an executable (DuckDB) SQL statement. After executing the statement, the result could be validated from the LLM. If the hypothesis proves valid, then the process stops. Otherwise, refined

[3] A textual description of the object-centric data structure in *pm4py* is available at `https://github.com/fit-alessandro-berti/pm-manuals-for-llms/blob/main/pm4py_manual.txt`.

[4] The documentation is available at the URL `https://pm4py.fit.fraunhofer.de/static/assets/api/2.7.11/index.html`

[5] A textual file collecting the APIs is available at the address `https://github.com/fit-alessandro-berti/pm-manuals-for-llms/blob/main/pm4py_manual.txt`.

[6] `https://chat.openai.com/g/g-EjtLrVyWH-pm4py-assistant`

[7] `https://github.com/pm4py/pm4py-core/blob/release/examples/llm/04_hypothesis_generation.py`

hypotheses may be provided by the LLM. The automatic formulation of hypotheses also suffers from LLM hallucinations [8] and, in a minor way, privacy issues due to the provision of some textual abstractions in the prompt requesting the generation of hypotheses.

4 Conclusion

The *pm4py* process mining library offers a rich integration of PM with LLM, implementing all the paradigms described in Section 2. Potential development directions, dependent on the future capabilities of LLMs/LVLMs include the extraction of event logs from videos, the generation of visual explanations for a business process, and autonomous process discovery/conformance checking from the event data.

References

1. Berti, A., Kourani, H., Hafke, H., Yun-Li, C., Schuster, D.: Evaluating Large Language Models in Process Mining: Capabilities, Benchmarks, Evaluation Strategies, and Future Challenges. In: Proceedings of the BPM-DS 2024 Working Conference (TBP). Springer (2024), `https://doi.org/10.48550/arXiv.2403.06749`
2. Berti, A., Schuster, D., van der Aalst, W.M.P.: Abstractions, scenarios, and prompt definitions for process mining with llms: A case study. In: Business Process Management Workshops - BPM 2023 International Workshops. Lecture Notes in Business Information Processing, vol. 492, pp. 427–439. Springer (2023)
3. Berti, A., van Zelst, S.J., Schuster, D.: Pm4py: A process mining library for python. Softw. Impacts **17**, 100556 (2023)
4. Jessen, U., Sroka, M., Fahland, D.: Chit-chat or deep talk: Prompt engineering for process mining. CoRR **abs/2307.09909** (2023), `https://doi.org/10.48550/arXiv.2307.09909`
5. Kourani, H., Berti, A., Schuster, D.: Process Modeling With Large Language Models. In: Proceedings of the EMMSAD 2024 Working Conference (TBP). Springer (2024), `https://doi.org/10.48550/arXiv.2403.06749`
6. Li, J., Hui, B., et al., G.Q.: Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In: NeurIPS 2023 (2023)
7. Maggi, F.M.: Declarative process mining with the declare component of prom. In: Proceedings of the BPM Demo sessions 2013. CEUR Workshop Proceedings, vol. 1021. CEUR-WS.org (2013)
8. Martino, A., Iannelli, M., Truong, C.: Knowledge injection to counter large language model (LLM) hallucination. In: ESWC 2023 Satellite Events. Lecture Notes in Computer Science, vol. 13998, pp. 182–185. Springer (2023)
9. Verbeek, H.M.W.: The log skeleton visualizer in prom 6.9. Int. J. Softw. Tools Technol. Transf. **24**(4), 549–561 (2022)
10. Yao, Y., Duan, J., et al., K.X.: A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. CoRR **abs/2312.02003** (2023), `https://doi.org/10.48550/arXiv.2312.02003`