Re-Thinking Process Mining in the AI-Based Agents Era

Alessandro Berti[1,2], Mayssa Maatallah[3], Urszula Jessen[4,5], Michal
Sroka[6], Sonia Ayachi Ghannouchi[3]

[1] Process and Data Science Chair, RWTH Aachen University, Aachen, Germany
[2] Fraunhofer FIT, Sankt Augustin, Germany
[3] Higher Institute of Management of Sousse, University of Sousse, Sousse, Tunisia
[4] Process Insights, ECE Group Services, Hamburg, Germany
[5] Eindhoven University of Technology, The Netherlands
[6] Microsoft
a.berti@pads.rwth-aachen.de;
{mayssamaatallah, ghannouchisonia.ayachi}@isgs.u-sousse.tn;
u.a.jessen@tue.nl; misroka@microsoft.com;

Abstract. Large Language Models (LLMs) have emerged as powerful
conversational interfaces, and their application in process mining (PM)
tasks has shown promising results. However, state-of-the-art LLMs strug-
gle with complex scenarios that demand advanced reasoning capabili-
ties. In the literature, two primary approaches have been proposed for
implementing PM using LLMs: providing textual insights based on a
textual abstraction of the process mining artifact, and generating code
executable on the original artifact. This paper proposes utilizing the AI-
Based Agents Workflow (AgWf) paradigm to enhance the effectiveness of
PM on LLMs. This approach allows for: i) the decomposition of complex
tasks into simpler workflows, and ii) the integration of deterministic tools
with the domain knowledge of LLMs. We examine various implementa-
tions of AgWf and the types of AI-based tasks involved. Additionally,
we discuss the CrewAI implementation framework and present examples
related to process mining.

Keywords: AI-Based Agents Workflow · Agents Crew · Process Mining
· Large Language Models

1   Introduction

Process Mining (PM) is a branch of data science aiming to infer process-related
insights starting from the event data recorded by the information systems sup-
porting the execution of the processes. Several types of techniques have been
proposed within process mining, including process discovery (the automated
discovery of a process model), conformance checking (comparing the behavior
of an event log against the process model), and predictive analytics (next activ-
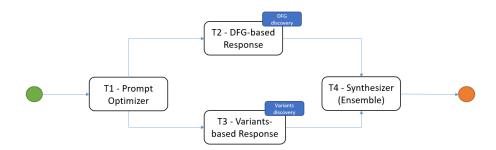ity/remaining time in a case).

Fig. 1: Example AgWf, reported from Fig. 2, with four tasks aimed to combine the DFG and variants abstraction of an event log to respond to the inquiry of the user, which is preliminarily optimized by another task.

Large Language Models (LLMs) have emerged as powerful PM assistants [3], being able to: 1) effectively respond to inquiries over a textual abstraction of a PM artifact (for example, identifying semantic anomalies or root causes); 2) produce code (like Python or SQL) that can be executed over a PM artifact.

However, the implementation paradigms 1) and 2) fail in more complex scenarios. For instance, composite tasks, which could be resolved by human analysts in different steps (for example, estimating the level of unfairness in an event log could be divided into i) identification of the protected group; ii) comparison between protected and non-protected group), are difficult for an LLM which could fail to decompose and execute each of them correctly [21]. Moreover, some tasks may require the production of code (to compute reliable statistics over the entire event log) but also the semantic capabilities of LLMs for the interpretation of the results obtained executing the code (for instance, a LLM might generate some declarative process model. Then, code is generated to apply conformance checking between the log and the process model. In the last step, the semantic understanding of the LLM may be required to interpret the results).

In this paper, we propose the application of the *AI-Based Agents Workflow (AgWf)* paradigm [10] in the PM context. AgWf(s) combine deterministic functions (called tools) and non-deterministic functions (AI-based tasks) to take the best of the two worlds: the rich set of process mining techniques has already been developed, as well as the semantic ability of LLMs. Moreover, AgWf(s) are deeply based on the *divide-et-impera* principle, in which difficult tasks are decomposed into manageable (by the LLM) units, with the overarching goal of increasing the quality of the overall result.

In the following, we analyze some PM applications benefitting from an AgWf-based implementation. Moreover, we introduce some types of AI-based tasks (routers, ensembles, evaluators, improvers) that are helpful to implement PM pipelines. We also present the CrewAI implementation framework which is helpful to implement AgWf(s), along with two examples tailored to the PM context. In Fig. 1, we see an example of multi-task AgWf.

The rest of the paper is organized as follows. In Section 2, we discuss the related work on PM-on-LLMs and scientific workflows in PM. In Section 3, we

present the definition of AgWf along with a running example, possible implementations, and types of tasks. In Section 4, we present the CrewAI framework for implementing AgWf(s). In Section 5, we discuss the future research directions in implementing AI-based agents. Eventually, Section 6 concludes the paper.

2   Related Work

Process Mining on LLMs: In [5], textual abstractions of PM artifacts are provided to the LLMs in order to respond to inquiries. On the other hand, the translation of the inquiry to SQL statements executable against the original data source is proposed in [7], along with error-correction mechanisms. In [3], some types of PM that could be implemented on LLMs (semantic anomaly detection, root cause analysis, visual recognition, fairness assessments) are discussed. Moreover, different implementation paradigms (*direct provision of insights*, *code generation*, and *hypotheses generation*) are proposed. In [2], a comprehensive benchmark for PM tasks on LLMs is proposed.
Connection to Traditional Process Mining Workflows: Scientific workflows, for example based on RapidMiner [11], Knime [8], or SLURM [15], have been used in PM to increase reproducibility and standardize large-scale experiments. However, the overarching goal of AgWf(s) is not to ensure reproducibility or standardization but to ensure the feasibility of the overall pipeline by adopting a *divide-et-impera* approach and *using the right tool for every task*. Moreover, AgWf(s) are non-deterministic by nature, while for most PM workflows, the same output would be obtained starting from the same inputs (determinism).

3   Methodology

3.1   AI-Based Agents Workflows

In Def. 1, we propose the definition of *AI-Based Agents Workflow (AgWf)*. The definition contains deterministic tools (F) transforming strings to other strings, and AI-based (non-deterministic) tasks (T). We indicate with the symbol $\rightsquigarrow$ a non-deterministic function, providing possibly different outputs for the same input. Each task in T may be associated with a set of tools (via the "tools" function). The selection of the tool for the purpose is also a non-deterministic (AI-based) function. We indicate with U_Σ the universe of all strings.

Definition 1 (AI-Based Agents Workflow (AgWf)).
*An AI-Based Agents Workflow (*AgWf*) is a tuple $(F, T, tools, selector, prec, t_1, t_f)$ for which:*

- $F \subseteq (U_\Sigma \nrightarrow U_\Sigma)$ *is a set of tools.*
- $T \subseteq (U_\Sigma \rightsquigarrow U_\Sigma)$ *is a set of (AI-based) tasks.*
- $tools : T \rightarrow \mathcal{P}(F)$ *associates a set of tools to a task.*
- $selector : U_\Sigma \times \mathcal{P}(F) \rightsquigarrow F$ *selects a tool (for the given inquiry) among the available ones.*

- $prec : T \rightarrow \mathcal{P}(T)$ *associates a task with a set of preceding tasks.*
- $t_1 \in T$ *is the initial task of the workflow.*
- $t_f \in T$ *is the final task of the workflow.*

In the definition, we never explicitly mention the term *agent*. We assume that the (AI-based) *agent* is the underlying executor of the (AI-based) tool. In particular, the (AI-based) agent is involved in the execution of two different non-deterministic activities:

- The selection of the tool, among the available ones for the task, to be used for the purpose (while the selected tool itself is deterministic).
- The execution of the task itself, which leads to the final response of the task.

Different (AI-based) tasks can be associated with different (AI-based) agents, accounting for the different types of skills of the agents. For example, simpler tasks could be executed with simpler agents (decreasing the costs and the execution times), while more complex tasks require competent agents.

We also note that in Def. 1, the tasks/tool selections are non-deterministic, but the definition of the workflow is static. In Section 5, we discuss the next natural step in the definition of AI-based agents: the automatic orchestration of the workflows (the tasks and their order are decided by an orchestrator).

In Def. 2, we define the *execution* of an AI-based workflow. First, a sequence of tasks respecting the provided order is extracted from the workflow. Then, each task is executed, leading to an output that is appended (via the $\oplus$ function) to the original inquiry. While the definition in Def. 2 could be modified to account for concurrent executions of the tasks, the currently available implementation framework (for AgWf) works stably within sequential executions.

Definition 2 (AgWf Execution - Sequential).
Let $AgWf = (F, T, tools, selector, prec, t_1, t_f)$ be an AI-based agents workflow. We define as execution a tuple $ExAgWf = (AgWf, S_T, S_\Sigma)$ such that:

- $S_T = \langle t_1, \ldots, t_f \rangle$, *with $t_1, \ldots, t_f \in T$, is a sequence of tasks respecting prec.*
- $S_\Sigma = \langle \sigma_0, \sigma_1, \ldots, \sigma_f \rangle$ *is a sequence of strings, with σ_0 being the initial state of the workflow (i.e., the initial inquiry of the user).*
- *For each $i \in \{1, \ldots, f\}$:*
    - *If $tools(t_i) = \emptyset$, $\sigma_i = \sigma_{i-1} \oplus t_i(\sigma_{i-1})$.*
    - *If $tools(t_i) \neq \emptyset$, $\sigma_i = \sigma_{i-1} \oplus t_i(\sigma_{i-1} \oplus selector(\sigma_{i-1}, tools(t_i))(\sigma_{i-1}))$.*

In Def 2, we separate between tasks without associated tools and tasks with associated tools. For the first type, the state of the workflow is defined as the concatenation of the previous state and of the result of the execution of the (AI-based) task. For the second type, we execute:

1. The selection of the tool, among the available ones.
2. The (deterministic) tool applied on the previous state.
3. The (AI-based) task is executed on the previous state and on the result of the application of the selected tool.

While the output of the deterministic tool is not persisted in Def. 2, it is actively used in determining the final answer of the (AI-based) task.

3.2   AgWf Running Example

In Fig. 2, we see an example of AI-based agents workflow, aiming to exploit two different abstractions (directly-follows graph and process variants) to respond to the inquiry of the user. The individual results are eventually merged by an ensemble, extracting the best of the single answers. The strategy, requiring the execution of four different prompts, potentially leads to better results since different abstractions represent different views on a given PM artifact. For example, in [4] it is discussed that the variants abstraction could be better suited for root cause analysis, while for semantic anomaly detection tasks, the knowledge of the directly-follows graph may be sufficient. However, in some instances, the opposite choice might be more effective. For instance, in a process in which the performance problems lay in a single transition between two activities, the DFG abstraction might be more effective in showing the root cause of the performance issue. On the other hand, in a P2P process, if an invoice is paid twice non-consecutively, that would be hidden in the directly-follows graph abstraction but would be visible in the variants abstraction.

Considering always Fig. 2, we see clear start/end tasks (**T1** and **T4**). Two different sequences of tasks are allowed by the workflow (since **T2** and **T3** are interleaved): $\langle \mathbf{T1}, \mathbf{T2}, \mathbf{T3}, \mathbf{T4} \rangle$ and $\langle \mathbf{T1}, \mathbf{T3}, \mathbf{T2}, \mathbf{T4} \rangle$. The execution of each task appends the result to the input string. For instance, an initial inquiry of the user, *Tell me the violations in the process contained in the event log at* /**home-/erik/p2p.xes**, can be optimized by **T1** appending *Could you analyze the behavior in the process, providing a list of anomalous behavior?*. Then **T2** would append its analysis based on the DFG, for instance, *"Create Purchase Requisition" should never transition to "Create Purchase Order" without approval.* **T3**, on the other hand, would exploit the behavior evidenced in the variants, appending *"You should never pay twice the same invoice"*. Eventually, **T4** would provide a composition of the two provided insights, appending *"In conclusion, the main problems are the lack of standardization in the management of purchase requisitions and multiple payments for the same invoice."*.
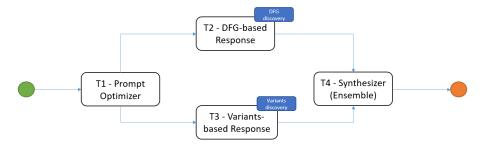

Fig. 2: Example AgWf in which the first task (**T1**) is optimizing the received inquiry, then two different tasks (**T2** and **T3**) are executed using two different textual abstractions (directly-follows graph and process variants) to retrieve an answer, and eventually the responses are synthesized by an ensemble (**T4**).

Always in Fig. 2, we see that **T2** and **T3** are annotated with some tools, i.e., *DFG discovery* and *Variants discovery*. For instance, these can be implemented using the *pm4py* process mining library [5], opening the XES referenced in the input string and applying a discovery operation. The results of the tools are appended to the prompt before the (AI-based) task is executed. Tools help to avoid "re-inventing the wheel", releasing the task from the duty to compute the DFG/process variants[7].

3.3   Possible Implementations

Several possible implementations of AgWf(s) are possible for the same tasks In Fig. 3, Fig. 4, and Fig. 5, we see different implementations of AgWf for the same problem (bias detection). However, arguably, there is a clear rank of effectiveness in the implementations.

The least effective implementation is represented in Fig. 3. There is a single task, whose final output should be an estimation of the unfairness level in the considered event log. The task would be resolved by a human analyst as follows [13]:

---

[7] Assuming that the provision of the entire event log in the prompt is feasible, which in many cases is not, due to the limited length of strings accepted by the AI-based tasks.

Fig. 3: Example AgWf for bias detection in process mining (single task - multiple tools).

Fig. 4: Example AgWf for bias detection in process mining (multiple tasks - multiple tools).
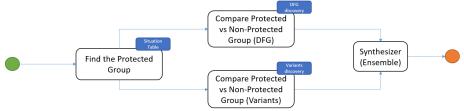

Fig. 5: Example AgWf for bias detection in process mining (multiple tasks - single tool per task).

- The event log is divided between cases belonging to the protected group and cases belonging to the non-protected group.
- The behavior is compared between the protected and non-protected groups.

However, AI-based tasks as in Definition 1 are allowed to use a single tool and produce a one-shot response to the provided inquiry. Therefore, they would not be able to perform successfully the pipeline. In the best case, they would be able to infer some fairness-related insights from the process variants (e.g., when activities potentially related to discriminations are contained in the process variants).

The workflow proposed in Fig. 4 is decomposed into two tasks, identification of the protected group and comparison between the protected versus the non-protected group. The pipeline is sound. However, the second task needs to make a choice between several tools, which could lead to incomplete fairness insights.

The workflow proposed in Fig. 5 is decomposed into four tasks. The second and third tasks compare the protected and non-protected groups in two different ways, based on the DFG and process variants abstractions. The results are then collated by the ensemble to form the final report on the unfairness contained in the event log.

Since current AI-based agents are still limited in their performance and scope of action, the decomposition of the final goal (shown in Fig. 3) into different sub-tasks, each limited in scope, leads more straightforwardly to the desired output.

3.4   Types of Tasks

In Fig. 2 and Fig. 5, we show the role of tasks in AgWf workflows. In this section, we aim to discuss different types of tasks and their utility, especially focusing on the PM context.

- *Prompt Optimizers*: tasks accepting the original inquiry of the user and transforming it to a language tailored to the capabilities of the AI agents. They usually are not associated with any tool, as their role is to optimize the cleanliness and effectiveness of the inquiry.
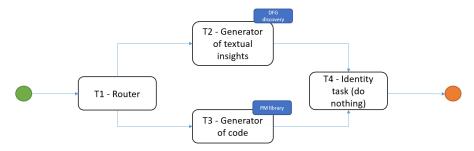

Fig. 6: An AgWf in which an inquiry is redirected either to **T2** (LLM-based textual insights, i.e., checking the semantics) or **T3** (production of code that is executed against the log in an existing PM library).

– *Ensembles*: tasks accepting a prompt containing a collation of insights (collected from different tasks offering a different perspective) and returning a coherent text containing the main results of the analysis. For example, the ensemble could summarize analyses over different dimensions (control-flow, temporal, data, resource) into a unified report on the process.
– *Routers*: tasks accepting a prompt and deciding which one of the depending nodes should be executed. While explicit routing is not allowed within the context of Def. 1, the following tasks could be instructed to consider the output of the routing node and possibly skip the production of further output. For example, in Fig. 6, we see a typical routing decision, i.e., choosing if the problem should be resolved directly by the LLM as it is a semantic task and/or it does not require extensive access to the attributes of the event log, or generating some code executable against the event log using a PM library such as *pm4py*.
– *Evaluators*: tasks evaluating the output of a previous task and assessing the quality, for instance, assigning a score between 1.0 and 10.0. This might help to understand the effectiveness of the task's execution. While the definition of *AgWf* does not allow for loops, in case of outputs with low quality is it possible to implement a "wrap back" mechanism in which the execution is taken back to a previous state and repeated.
– *Output Improvers*: trying to enhance the quality of the output of the previous tasks. For instance, the insights could be refined ("second opinion") or, in the case of code generation, the quality or security of the code can be improved.

4   Implementation Framework

In this section, we present the *CrewAI* Python framework `https://github.com/crewAIInc/crewAI` for the implementation of AgWf(s) on top of Large Language Models. It is based on the following concepts:

– *AI-based agents* are defined as LLMs plus system prompts. The system prompt tailors the behavior of a given LLM to a given role (*role prompting* [18]).
– *AI-based tasks* are defined based on a textual instruction. They are associated with an AI-based agent.
– *Tools* are defined as Python units (classes/functions). A task can be connected to some tools. The selection operates on the documentation string of the different tools, including the input arguments and the type of the output.
– In the traditional implementation, a sequential order of execution for the tasks is defined. More recently, a concept of concurrent execution (*hierarchical processes*) has been tried, but further work is needed.

An important criteria for the selection of the LLM is its ability as selector for the most suitable tool. Notably, LLMs such as *Llama-3.1*[8], *Qwen 2.0*[9], *Mistral Large*

---

[8] `https://ai.meta.com/blog/meta-llama-3-1/`

[9] `https://medium.com/@smalltong02/the-comprehensive-evaluation-of-the-agency-capability-of-the-qwen-2-model-cb7eb675c091`

2[10] or *GPT-4O/GPT-4O-Mini*[11] offer excellent support to implement AgWf. Also, since a workflow contains potentially many different tasks, the speed of the model is important. For instance, *Llama 3.1 70B* and *GPT-4O-Mini* could be a preferred choice over their bigger siblings *Llama 3.1 405B* and *GPT-4O* due to their satisfactory performance at a lower computational price.

CrewAI supports also additional concepts in comparison to Def. 1:

– The *entity memory* is a dictionary persisting variables produced/accessed during the execution of the workflow. For instance, from an initial log **log**, we could create two sub-logs (for example, **training** and **test**). The **training** log could be then accessed to generate some hypotheses that are then tested on the **test** log.
– Python functions (*callbacks*) could be called at the end of the execution of some tasks (for example, to persist the result, or check the formal correctness).

In the following, we will propose two examples of AI-based workflows. We implemented in CrewAI the fairness workflow shown in Fig. 4. A Jupyter notebook is available at the address `https://github.com/fit-alessandro-berti/agents-trial/blob/main/02_fairness_assessment.ipynb`. The selected LLM for the task is *Qwen 2.0 72B*. In particular, the first task (identification of the protected group) generates some code (SQL statement) executed against the event log in order to split the behavior between "protected" and "non-protected" cases. Using the advanced features provided by CrewAI, both event logs are stored in the entity memory for usage in the following task. The following task (comparison between protected and non-protected groups) calls the computation of the DFG on both event logs and returns a textual list of insights. We see that we assign each task to a different *agent*. Despite both agents are being supported by the same LLM (*Qwen 2.0 72B*), the system prompt defines a different purpose for each LLM. The tasks are defined, following the CrewAI framework implementation, with a *description* and an *expected output*.

We also define another AgWf for root cause analysis, including two different mechanisms of evaluation, at the address
`https://github.com/fit-alessandro-berti/agents-trial/blob/main/01_root_cause_analysis_insights.ipynb`. The workflow is shown in Fig. 7. In particular, the first step **T1** performs root cause analysis starting from the DFG abstraction, producing a list of potential root causes. Then, **T2** assigns to each insight a confidence score [9] from 1.0 (minimum) to 10.0 (maximum). **T3** needs to provide the chain-of-thought [19] for the first of the provided insights, so the detailed reasoning steps. The graded insights and the detailed reasoning steps (for the first insight) are then returned at the end of the workflow. We see that we also define three different agents on the same LLM with different system

---

[10] `https://techcommunity.microsoft.com/t5/ai-machine-learning-blog/ai-innovation-continues-introducing-mistral-large-2-and-mistral/ba-p/4200181`

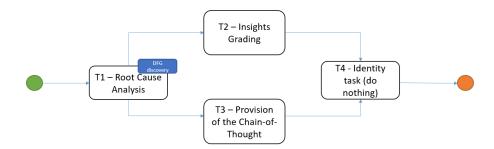[11] `https://cobusgreyling.medium.com/langchain-based-plan-execute-ai-agent-with-gpt-4o-mini-243ee57c6a5a`

Fig. 7: Workflow for root cause analysis, including the insights grading and the provision of the chain-of-thought as evaluation steps.

prompts. In comparison to the previously defined workflow, the tasks are easier to perform. Therefore, we propose to use the *Qwen 2.0 8B* LLM, which is significantly smaller than *Qwen 2.0 72B* LLM.

5   Next Steps

Automatic Definition of AgWf(s): In the previous sections, we show how tasks could be decomposed into an AgWf. However, decomposition is done by humans. Some approaches [14,17] show that the same decomposition task could be performed by an "orchestrating" LLM. In particular, the original task is decomposed into a sequence of smaller tasks assigned to specialized agents. One of the main challenges identified in [14] is the comprehension of the original task. In particular, it is argued that the orchestrating LLM should be instructed to request clarifications on the task.

Tasks Keeping the Human-in-the-Loop: AgWf(s) could be used to automate many tasks. However, the execution of some tasks might benefit from clarifications provided by the end user [1]. For instance, the prompt optimizer depicted in Fig. 2 could struggle to optimize a very generic inquiry (such as "What are the problems in the process?"), and could benefit from additional clarifications provided by the user.

Evaluating AgWf(s): In this paper, we argue that AgWf(s) are useful tools to increase the quality of the output on a specific task by decomposing it into smaller tasks executed by specialized agents. The assessment of LLM-based outputs is challenging, with the *LLMs-as-a-Judge* paradigm [6] being a popular solution. Within AgWf(s), the overall effectiveness (quality of the final output) depends on the effectiveness of the single agents. For instance, errors in the initial routing of the inquiry can result in a significantly lower quality output even if all the other tasks are performed optimally. Therefore, we argue that the quality of the output of the single tasks should be assessed.

Also, when multiple agents are involved, the collaboration and psychological traits should be considered [20,16]. Tasks could be implemented with self-awareness or also awareness of the context (overarching goal/workflow context).

In [12], it is argued that even currently top-performing LLMs show poor cooperation behavior/negotiation skills, leading sometimes to poor outcomes.

Maturity of the Tool Support: several frameworks are proposed to implement AgWf(s). In particular, we need to mention the *LangGraph* framework `https://langchain-ai.github.io/langgraph/`, which offers comprehensive support for tasks/tools (for instance, direct connection to the search engines is provided). However, due to the ever-evolving structure of the library, prototypes depend on a specific version of the library and can stop working on a future version. Also, some user interfaces have been proposed for LangGraph[12], but they are also highly dependent on the version of the underlying library. *CrewAI* has been proposed in the paper as a "compromise" solution between ease-of-use and features support. However, it still lacks some advanced features offered by LangGraph and lacks a graphical interface. The *AutoGen* solution by Microsoft `https://microsoft.github.io/autogen/` is also a reasonable choice but lacks the completeness of LangGraph. It comes with a user-friendly graphical interface allowing for the definition of the workflows. Overall, all the considered solutions need further work to reach high maturity.

6   Conclusion

In this paper, we analyzed the limitations of the currently proposed implementations for PM-on-LLMs, proposing AgWf(s) as a possible solution involving i) the decomposition of the original task in smaller units; ii) the combination between AI-based task execution and "deterministic" tools (for instance, using the features offered by process mining libraries). AgWf(s) have a different goal (i.e., maximizing the quality of the output) than previously proposed scientific workflows (i.e., allowing the reproducibility of scientific experiments). We propose different types of AI-based tasks useful for process mining applications, including prompt optimizers, ensembles, routers, evaluations, and output improvers. We use the *CrewAI* framework to implement some example AgWf(s) (including root cause analysis and bias detection in process mining event logs). We also discuss future directions for research and development, including the automatic definition of the workflows, evaluation frameworks for agents, and increased maturity of the underlying frameworks/tool support. Overall, AgWf(s) offer a powerful tool for PM-on-LLMs, requiring a *divide-et-impera* mindset.

References

1. Abuelsaad, T., Akkil, D., Dey, P., Jagmohan, A., Vempaty, A., Kokku, R.: Agent-e: From autonomous web navigation to foundational design principles in agentic systems. arXiv preprint arXiv:2407.13032 (2024)
2. Berti, A., Kourani, H., van der Aalst, W.M.: Pm-llm-benchmark: Evaluating large language models on process mining tasks (2024)

---

[12] For example, `https://github.com/LangGraph-GUI/LangGraph-GUI`

3. 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. Springer (2024)
4. Berti, A., Schuster, D., van der Aalst, W.M.P.: Abstractions, scenarios, and prompt definitions for process mining with llms: A case study. In: BPM 2023 Workshops. vol. 492, pp. 427–439. Springer (2023)
5. Berti, A., van Zelst, S.J., Schuster, D.: Pm4py: A process mining library for python. Softw. Impacts **17**, 100556 (2023)
6. Huang, H., Qu, Y., Liu, J., Yang, M., Zhao, T.: An empirical study of llm-as-a-judge for LLM evaluation: Fine-tuned judge models are task-specific classifiers (2024)
7. Jessen, U., Sroka, M., Fahland, D.: Chit-chat or deep talk: Prompt engineering for process mining (2023)
8. Kourani, H., van Zelst, S.J., Lehmann, B., Einsdorf, G., Helfrich, S., Liße, F.: PM4KNIME: process mining meets the KNIME analytics platform (extended abstract). In: ICPM 2022 Demo Track. CEUR Workshop Proceedings, vol. 3299, pp. 65–69. CEUR-WS.org (2022)
9. Li, L., Chen, G., Su, Y., Chen, Z., Zhang, Y., Xing, E.P., Zhang, K.: Confidence matters: Revisiting intrinsic self-correction capabilities of large language models (2024)
10. Li, Y., Zhang, Y., Sun, L.: Metaagents: Simulating interactions of human behaviors for llm-based task-oriented coordination via collaborative generative agents (2023)
11. Mans, R., van der Aalst, W.M.P., Verbeek, H.M.W.: Supporting process mining workflows with rapidprom. In: BPM 2014 Demo Track. CEUR Workshop Proceedings, vol. 1295, p. 56. CEUR-WS.org (2014)
12. Piatti, G., Jin, Z., Kleiman-Weiner, M., Schölkopf, B., Sachan, M., Mihalcea, R.: Cooperate or collapse: Emergence of sustainability behaviors in a society of LLM agents (2024)
13. Pohl, T., Berti, A., Qafari, M.S., van der Aalst, W.M.P.: A collection of simulated event logs for fairness assessment in process mining. In: BPM 2023 Demo Track. CEUR Workshop Proceedings, vol. 3469, pp. 87–91. CEUR-WS.org (2023)
14. Rasal, S., Hauer, E.J.: Navigating complexity: Orchestrated problem solving with multi-agent llms (2024)
15. Sadeghibogar, Z., Berti, A., Pegoraro, M., van der Aalst, W.M.P.: Applying process mining on scientific workflows: a case study (2023)
16. Sreedhar, K., Chilton, L.B.: Simulating human strategic behavior: Comparing single and multi-agent llms (2024)
17. Wang, H., Li, T., Deng, Z., Roth, D., Li, Y.: Devil's advocate: Anticipatory reflection for LLM agents (2024)
18. Wang, R., Mi, F., Chen, Y., Xue, B., Wang, H., Zhu, Q., Wong, K., Xu, R.: Role prompting guided domain adaptation with general capability preserve for large language models (2024)
19. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: NeurIPS 2022 (2022)
20. Zhang, J., Xu, X., Zhang, N., Liu, R., Hooi, B., Deng, S.: Exploring collaboration mechanisms for LLM agents: A social psychology view (2023)
21. Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q.V., Chi, E.H.: Least-to-most prompting enables complex reasoning in large language models. In: ICLR 2023. OpenReview.net (2023)