Event Log Extraction for Process Mining Using Large Language Models*

Vinicius Stein Dani[1], Marcus Dees[2], Henrik Leopold[3], Kiran Busch[3], Iris Beerepoot[1], Jan Martijn E. M. van der Werf[1], and Hajo A. Reijers[1]

[1] Utrecht University,
Princetonplein 5, 3584 CC, Utrecht, The Netherlands
{v.steindani,i.m.beerepoot,j.m.e.m.vanderwerf,h.a.reijers}@uu.nl
[2] Uitvoeringsinstituut Werknemersverzekeringen,
Amsterdam, The Netherlands
marcus.dees@uwv.nl
[3] Kühne Logistics University,
Großer Grasbrook 17, 20457 Hamburg, Germany
{henrik.leopold,kiran.busch}@the-klu.org

Abstract. Process mining is a discipline that enables organizations to discover and analyze their work processes. A prerequisite for conducting a process mining initiative is the so-called event log, which is not always readily available. In such cases, extracting an event log involves various time-consuming tasks, such as creating tailor-made structured query language (SQL) scripts to extract an event log from a relational database. With this work, we investigate the use of large language models (LLMs) to support event log extraction, particularly by leveraging LLMs ability to produce SQL scripts. In this paper, we report on how effectively an LLM can assist with event log extraction for process mining. Despite the intrinsic non-deterministic nature of LLMs, our results show the potential of future LLM-assisted event log extraction tools, especially when domain and data knowledge are available. The implementation of such tools can increase access to event log extraction to a broader range of users within an organization by reducing the reliance on specialized technical skills for producing relational database query scripts and minimizing manual effort.

Keywords: Process mining · Event log extraction · Relational database · Prompt engineering · Large language model · LLM · Assistance.

1 Introduction

Process mining is a discipline that helps organizations to discover and analyze their work processes [2]. A prerequisite for conducting any process mining initiative is the so-called event log, which is not always readily available. When it is

* Accepted manuscript on September 25, 2024, to the 30th International Conference on Cooperative Information Systems (CoopIS).

not, extracting an event log involves laborious, time-consuming tasks [19]. The event log extraction is particularly challenging when an organization needs to extract event logs for continuously changing purposes. For instance, new demands emerging from various departments can make the replication of event log extraction strategies impractical. In such a situation, new database query scripts must be designed for each demand, which further increases the time needed for event log extraction. As a result, organizations have to wait longer to gain insights from their data.

Large language models (LLMs) emerged as tools capable of analyzing and generating text, offering potential applications in different domains and research fields [11,16,21]. These models have demonstrated capabilities in natural language processing tasks, including text generation, translation, and summarization [25,26]. In the context of process mining, studies have shown the potential and applicability of LLMs [3,12]. For example, in [3] the authors shown the applicability of LLMs to process analysis-related tasks, while in [12] the authors have demonstrated how LLMs can be used for assisting on process modeling tasks. Moreover, different studies experimented with using LLMs to map natural language questions on a given relational database into ready-to-use SQL queries [9]. Considering this, it may be beneficial to use LLMs for event log extraction, what has not been explored before.

With this in mind, we argue that SQL query generation from natural language text using LLMs can potentially reduce the manual effort required for event log extraction. We hypothesize that this automation can streamline the event log extraction process by reducing the dependency on specialized technical skills from business analysts, making event log extraction more accessible to a broader range of users within an organization. Although LLMs are non-deterministic, their flexibility enables a quick adaptation to the continuously changing scenarios of different demands coming from a variety of departments. We argue that as new data-related requirements emerge, these models can quickly generate the appropriate relational database queries for event log extraction without the need for major reprogramming efforts. By integrating LLMs into the event log extraction workflow, having human-computer collaboration, organizations can increase their ability to quickly extract event logs, thereby gaining timely insights and getting more quickly to their decision-making moments based on the discovered process.

Against this background, we develop this work to answer the following research question: "*How effective are LLMs in assisting the event log extraction from relational databases?*" For this purpose, we assess state-of-the-art LLMs' performance against a number of structurally different relational databases and prompt engineering strategies. To do so, we leverage an experimental setup that executes each available prompt against each available database, extracts the LLM-generated SQL script from the LLM response, executes the generated SQL script against the database, and compares the LLM-assisted generated event log against a gold standard, outputting similarity scores by means of precision, recall, and F1-score. Despite the intrinsic non-deterministic nature of LLMs,

our results show that LLMs (particularly GPT-4o) can effectively assist in the event log extraction from relational databases when domain and data knowledge are available.

The remainder of this paper is structured as follows. In Section 2, we discuss the background of this work. Then, in Section 3, we present our experimental setup. In Section 4, we report on the acquired results of this work, while in Section 5 we provide a discussion and recommendations based on our findings. Finally, Section 6 concludes the paper.

2 Background

In this section, we first provide a brief introduction into process mining. Then, we discuss LLMs and prompt engineering strategies.

2.1 Process Mining

Process mining is a discipline that enables organizations to discover and analyze their work processes. This is possible by leveraging the available data in an event log format. In its simplest form, an event log is a comma-separated file containing a list of events with at least the following three attributes: CaseID, ActivityID, and timestamp. The CaseID represents a process instance, i.e., a particular trace of the execution of a process; the ActivityID represents an activity that was performed in the context of a particular trace or multiple traces; and the timestamp represents the exact moment in which an activity was performed [2,14]. For example, consider Table 1. In this table, we have two cases, A and B, of a purchase-to-pay process. In case A, three activities happened in the following temporal order "Create purchase order", "Receive goods", and "Pay invoice". In case B, there are only two activities: "Create purchase order", and "Pay invoice". In this example, one can see that in case B, the goods were not received, and a root-cause analysis can be conducted to understand whether this is an expected behavior or not.

What is important to note is that extracting such an event log often requires a significant amount of manual effort [19]. It typically involves an iterative process, in which SQL scripts are developed to integrate all required data in the target event log.

2.2 Large Language Models and Prompt Engineering

In recent years we have seen the insurgence of a variety of LLMs. Some are advertised as open-source, without actually sharing training data [10,22]. Others as proprietary, and claim that the data they used to train their models are public, which is not necessarily always the case [8,17,18]. What is a commonality for all widely available –publicly or privately– LLMs, is their intrinsic non-deterministic nature. A number of approaches to improve the effectiveness of LLMs have been produced in the form of prompt engineering strategies. For

example, in [23] the authors propose a prompt engineering strategy titled *chain-of-thought*, which breaks down tasks into intermediate steps. In [27], the authors propose a prompt strategy titled *tree-of-thought*, which explores multiple reasoning paths to respond to a prompt, potentially leading to more robust and creative solutions. In [4], the authors introduce the *few-shot examples* prompt strategy, which uses examples of how a desired task should be performed. In addition, in [24] the authors present a prompt engineering strategy titled *persona*, which elicits how to ask an LLM to behave in a certain way (e.g., as a process mining expert) while answering to the input prompt. In this work, we leverage these prompt engineering strategies.

A variety of studies have shown the applicability of LLMs in a number of domains and purposes [5,11,16,21], including SQL script generation [9] and process mining [3,12]. To the best of our knowledge, what has not been explored is the applicability of LLMs in assisting the event log extraction. With this work we provide a robust and methodological assessment of the performance of LLMs in assisting the event log extraction from relational databases.

3 Experimental Setup

In this section, we describe our experimental setup to assess how effective LLMs can be in assisting the event log extraction from relational databases. To this end, in Section 3.1, we provide an overview of our approach. In Section 3.2, we present the materials used in this work, which are composed by a number of LLMs, databases, prompts, and gold standard event logs. Finally, in Section 3.3 we elaborate on the analysis of the results, i.e., the comparison between a gold standard event log and its generated counterpart.

3.1 Overview

Figure 1 depicts the architecture of our experimental setup, which expects the following input objects: (i) an *LLM* to be evaluated; (ii) a *database* from which an event log can be extracted; (iii) a *prompt engineering strategy* to instruct an LLM about the event log extraction task; (iv) a *gold standard* event log to serve as a benchmark for comparison against the LLM-assisted generated event

Table 1: Example event log.

CaseID	**ActivityID**	**Timestamp**
A	Create purchase order	2023-10-26 10:17:33
B	Create purchase order	2023-10-27 06:03:19
A	Receive goods	2023-10-29 14:36:01
A	Pay invoice	2023-10-30 19:03:26
B	Pay invoice	2023-11-21 10:14:59

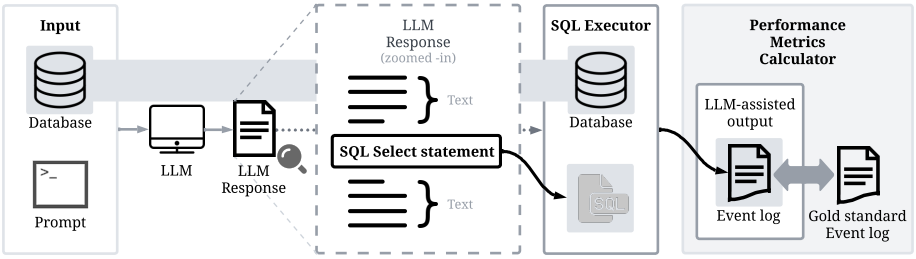


Fig. 1: Overview of our approach.

log. Upon receiving the expected input objects, our architecture automatically proceeds as follows: on the given *LLM*, it executes the *prompt*, appending to the prompt the *database* structure, and a request to generate an SQL script as an output. Then, the LLM-generated SQL script is extracted from the LLM response and executed against the database to produce the LLM-assisted event log generation. Next, the LLM-assisted generated event log is compared against the gold standard event log for the particular database, and similarity metrics are calculated. Finally, results can be manually analyzed and reported.

3.2 Materials

We define an architecture that handles four main input-objects: (i) a collection of different *LLMs* to be evaluated; (ii) a variety of *databases* from which event logs can be extracted; (iii) different *prompt engineering strategies* to instruct an LLM about the event log extraction task; and, (iv) *gold standard* event logs, which are previously known event logs extracted from each relational database to serve as a benchmark for comparison against the LLM-assisted generated event log. Next, we present further details about the materials used in this work[4].

LLMs. In this work, we evaluated four different OpenAI models: GPT 3.5-turbo, 4, 4-turbo, and 4o. We chose OpenAI models because they have demonstrated capabilities in natural language processing and code generation-related tasks [9,25,26], and because of its wide adoption [20].

An important aspect to have in mind regarding GPT is that OpenAI enables users of their LLM API to reduce the intrinsic non-deterministic nature of their models. This can be done by setting a feature called *temperature*[5] to zero. To minimize the randomness of the LLM responses we did so. Still, we learned that the LLM did not always produce the same output given the same input. We executed each test case (i.e., the combination of a prompt with a particular database) three times, averaging the results.

Databases. For the database selection, we aimed at choosing databases featuring diverse foreign key relations among the tables, as well as varied event

[4] https://github.com/KiriBu10/event-log-extraction-for-process-mining
[5] https://platform.openai.com/docs/guides/text-generation/

identification types, such as one event per row versus multiple events per row. In addition, we also aimed at databases with different sizes in terms of the number of tables and columns. What is more, we created two simplified versions for each selected database. We did so to enhance the generalizability of our findings, thus making our work more reliable in terms of our performance assessment of LLM-assisted event log extraction. The database selection also considered the prompt size limitations of the LLMs. For this work, we selected four databases: a synthetic purchase-to-pay (P2P) database, a real ERP system database [15], the BPI2016 Challenge database [6], and a real-life database from UWV (the Dutch Employee Insurance Agency). Table 2 outlines key characteristics of these selected databases. Versions 1 and 2 of each database are simplified versions of their version 3. Next, we delve into further details about each selected database:

– *P2P.* A synthetic purchase-to-pay database, inspired by [15], to serve as a basis for initial assessment of our experimental setup. This database particularly contributed to our understanding of the nuances in the differences between the LLM-assisted generated event log and the gold standard. It contains high-level information about orders, invoices, payments and shipment, and enables the event data understanding under two case notions related to the order and the invoice. Creating an event log for this database is straightforward due to the presence of only one-to-many foreign key relationships among the tables. Also, each table holds one event per record. Thus, each table represents events related to one particular ActivityID. The ActivityID of an event is therefore defined as the table name. In the most complex ver-

Table 2: Characteristics of the databases used in this work.

Database	**Version**	**# Tables**	**# Columns**	**# Case notions**	**# Timestamps**	**# Activity IDs**	**# Events per gold standard**	**Has foreign keys**
P2P	V1	2	5	2	1	0	8	Yes
	V2	4	11	2	1	0	13	Yes
	V3	6	19	2	1	0	13	Yes
ERP	V1	3	13	3	1	0	6	Yes
	V2	5	18	2	1	0	5	Yes
	V3	9	32	3	1	0	12	Yes
BPI2016	V1	3	42	2	>1	>1	30	No
	V2	2	35	2	1	>1	250	Yes
	V3	5	77	2	>1	>1	280	Yes
UWV	V1	2	7	1	1	1	11	No
	V2	3	17	2	>1	1	33	No
	V3	5	23	2	>1	1	43	No

sion of the database, version V3, also two tables are present containing clicks and configurations that have no relation with any of the other tables. These tables are added to observe how the LLM deals with unrelated information that is given as input.

- *ERP.* A real ERP system database fragment, as first used by Li et al. [15]. This database holds more information about orders, invoices, payments, and shipments, such as order items, shipment items, and many-to-many relations between order and invoice. In addition, this database has information about the customer. Thus, this database has three possible case notions, related to order, invoice, and customer. Creating an event log for this database is equivalent to the P2P database, with the difference that it holds many-to-many foreign key relationships among the tables. The ActivityID is determined by the table name.
- *BPI2016.* A database based on the BPI Challenge of 2016 [6]. In the BPI Challenge of 2016 a dataset was used with several types of customer contact data from UWV. The database contains clicks on the UWV website from logged-in users and of not logged-in users, messages sent through the website, calls to the call center and complaints from customers. This database has two case notions, i.e., website sessions with a SessionID and the customers with a CustomerID. Creating an event log for this database poses a different challenge when compared to the previous ones, as there are tables with multiple events per record. In this case, the ActivityID is determined by the column names instead of the table name.
- *UWV.* The UWV database is taken from a typical claim handling process executed by UWV. It combines tables from different information systems used in the execution of the process. It contains a table with letters sent to customers, a table with calls to the customer, a table with calls from customers and a table with timestamps per step in the claim handling process. The two case notions in the database are the customer and the claim handling session, i.e., one customer can have multiple claim handling sessions. Creating an event log for this database poses different challenges when compared to the previous databases, as there are no explicit foreign key relationships among the tables and there are tables with multiple events per record.

Prompts. Inspired by the rise of a variety of prompt engineering strategies, Table 3 shows seven prompts used in this work. The first five prompts are prompt strategies inspired by literature [4,23,24,27]. The sixth and seventh prompts, are prompts collaboratively engineered by the authors of this study. The sixth prompt leverages process mining and event log extraction-related knowledge in an attempt to improve the LLM-based event log extraction assistance. The seventh prompt leverages database-specific knowledge. On top of these prompt strategies, a number of statements are always appended to each prompt in the context of this work, in order to fit our automated experimental setup. These statements form the *Baseline* prompt, and include statements such as "*write an SQL statement with the columns: CaseID, ActivityID and timestamp*" and

Table 3: Examples of prompts used in this work. Prompts 1 to 5 are based on prompt engineering strategies from literature. Prompts 6 and 7 are based on process mining, and domain and data knowledge, respectively.

ID	Prompt
1	**Baseline** This prompt consists of the database schema and the prompt elements "write a SQL statement with the columns: CaseID, ActivityID and timestamp" and "return only the complete SQL query". All other prompts used in this work consist of the Baseline prompt, expanded with another particular prompt.
2	**Persona** [24] Act as a Process Mining Specialist that is an expert in event log extraction.
3	**Few-shot example** [4] Consider the following example of how an event log looks like when extracted from a database with two tables: {Table A} {Table B} {Extracted event log}
4	**Chain-of-thought** [23] Take a deep breath and think step-by-step in silence.
5	**Tree-of-thought** [27] Consider three experts are collaboratively answering a request using a tree-of-thoughts method. Each one of the three experts will share their thought process in detail, taking into account the previous thoughts of others and admitting any errors. They will iteratively refine and expand upon each others' ideas, giving credit where it's due. The process continues until a conclusive answer to the request is found. The request is the following.
6	**Process mining knowledge.** [Note: *This prompt is built using knowledge about process mining and event log-related concepts such as case notion, activity labels, and timestamps.*]
7	**Custom-made.** [Note: *For each database version, a fully customized prompt is created using domain and data knowledge.*]

"*return only the complete SQL query, nothing else should be part of the response*". Apart from these statements, we conducted several experiments to test different database-specific statements, such as "*each record in each table represents at least one event*" and "*if a table contains multiple columns containing a datetime format, then each of these datetime values is an event*", for the UWV database; or, "*when a table does not have a column that contains the selected case notion, combine the necessary tables to obtain this case notion*", for the BPI2016 database.

Gold standards. In the context of this work, the *gold standard* is an event log previously known to contain all the process instances of a particular case notion. Two researchers were involved in manually creating the gold standards separately. They wrote the SQL queries to extract the gold standard event log for each database, and assessed each others work with the help of a third researcher.

To ensure the gold standards are created in a similar fashion, we first determined a common strategy. For example, we assume that each record in a

table is an event, except when multiple timestamps are present. In that case each timestamp in a record defines an event. Secondly, when there is no column present that could serve as the ActivityID, then the name of the table is used as the ActivityID. When multiple timestamps exist in a table, we use the name of the table concatenated with the name of each respective timestamp column as ActivityID. We added elements of our gold standard construction strategy to the *process mining knowledge* prompt (prompt 6).

3.3 Performance Assessment

To analyse and assess the performance of the LLM-assisted generated event log, we compare the responses for different combinations of database and prompt against the gold standard defined for the database. In this work, we use three performance metric calculation (PMC) strategies:

PMC$_1$. Inspired by [1], we calculate the precision, recall, and F1-score by directly comparing the LLM-extracted event log L against the gold standard event log G. We identify true positives (TP), false positives (FP), and false negatives (FN) as follows: TP as $G \cap L$, FP as $L \setminus TP$, and FN as $G \setminus TP$. In the context of this work, a TP is defined as an event in L that also occurs in G, with exactly the same CaseID, ActivityID, and timestamp.

PMC$_2$. Inspired by our observation that often when the F1-score is zero in preliminary results using PMC$_1$, which is caused by ActivityIDs that do not match, we devised PMC$_2$. In PMC$_2$, we calculate the relaxed F1-score considering a partial match between an event in L and an event in G, ignoring the ActivityID. For example, consider an event as a triple (CaseID, ActivityID, and timestamp), event l_i as (1001, "Create order", 2024-06-02 12:37:40) in L, and event g_i as (1001, "order", 2024-06-02 12:37:40) in G. In the relaxed F1-score calculation, we accept l_i as equal to g_i. Note that the number of events in the comparison of the generated event log with the gold standard event log can become smaller when the ActivityID is ignored. Sometimes the ActivityID is the only distinguishing element between two or more events. This effect is due to the set abstraction we use in the comparison and has an impact on the calculation of the F1-score since the number of FNs can decrease.

PMC$_3$. In further observations we identified that often when the F1-score is zero for PMC$_1$, this is caused by ActivityIDs that do not perfectly match syntactically, but should do so semantically. We used the Levenshtein distance [13] to calculate the minimum number of edits necessary to transform the event from L to an event in G. For each event, all elements are first concatenated, e.g., the event (1001, "Create order", 2024-06-02 12:37:40) from L becomes "1001;Create order;2024-06-02 12:37:40". This transformation is also done for all events from G. Next, for each event in L the distance to each event in G is calculated. The similarity is expressed as 1 minus the ratio between the number of edits and the length of the longest of the two strings that are being compared, i.e., the gold standard value and the generated value. If the similarity value is above a threshold value of 0.75, we consider the match a TP; otherwise, it is an FP. The number of

*FP*s is calculated based on the size of the larger of the two event logs minus the number of *TP*s. Inspired by [7], we selected the 0.75 threshold experimentally. We manually identified that the balance between semantically similar words was higher for the 0.75 thresholds than, for example, when using 0.8, which means that if we had used 0.8 as a threshold, we would not have classified as *TP* events that should have been classified as such.

4 Results

In this section, we discuss the results of our experiments. Table 4 provides an overview of the results[6]. We show the results for GPT 4o, which was overall the best performing LLM used in this work. In total, we ran 252 API calls, as we had 84 test cases (four databases times three versions per database times seven prompts) that were executed three times each. In 25 out of 84 test cases (29.8%), the LLM produced different results in the three runs. Differences between the runs for a test case could be either small, indicating that only some of the events have changed in the result, or also large. In Table 4 we provide the average scores.

In the following sections, we report on the results of our experiments for each specific database, providing detailed findings that form the basis for our discussion section. Mind that whenever we say *generated event log* we refer to the *LLM-assisted generated event log*.

4.1 P2P Database

The P2P database is a synthetic database which incorporates two case notions (order and invoice). A key challenge for the LLM regarding this database is to select one case notion and write an appropriate SQL query. In the *custom-made* prompt (prompt 7) we specify a case notion, but for all other prompts the LLM chooses one. Figure 2 shows the F1-scores for three versions of the P2P database. For PMC_1 (cf., Figure 2a), prompt 7 is the only one reaching an F1-score of 1, meaning only the *custom-made* prompt performs well. The other prompts perform poorly, with prompts 3, 4, and 6 reaching F1-scores between 0.4 and 0.8 and prompts 1, 2, and 5 doing the worst at zero. For PMC_2 (cf., Figure 2b), the LLM is performing much better, with perfect F1-scores for V1 and scores for V2 generally varying between 0.8 and 1.0 (with the exception of prompt 5). V3 does not perform as well, with F1-scores generally between 0.4 and 0.7, but with a perfect score for prompt 7. For PMC_3 (cf., Figure 2c), all F1-scores are equal or higher than the F1-scores of PMC_2, except for the *process mining knowledge* prompt (prompt 6). Again, F1-scores are highest for V1, followed by V2, and then V3.

For PMC_1, all prompts but the custom-made prompt perform badly due to the LLM adding "_created" to the ActivityID (e.g., the LLM generates an ActivityID "order_created" instead of "order"). Note that we did not ask the LLM

[6] We focus on the F1-scores since precision and recall were very close in most cases.

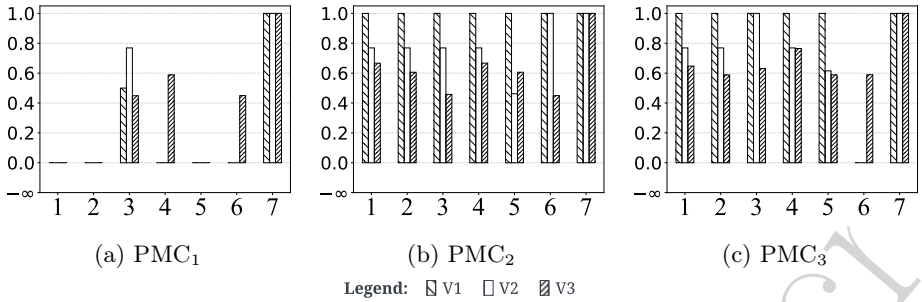
Table 4: Evaluation summary of database versions, with F1-scores for the three different Performance Metric Calculations (PMC). The $-\infty$ symbol is used when no valid event log was returned by the LLM. The greyscale cell-background goes from black (F1-score equals to zero) to white (F1-score equals to one).

Database	Prompt	V1			V2			V3		
		PMC_1	PMC_2	PMC_3	PMC_1	PMC_2	PMC_3	PMC_1	PMC_2	PMC_3
P2P	1	0.000	1.000	1.000	0.000	0.769	0.769	0.000	0.667	0.647
	2	0.000	1.000	1.000	0.000	0.769	0.769	0.000	0.606	0.588
	3	0.500	1.000	1.000	0.769	0.769	1.000	0.448	0.458	0.631
	4	0.000	1.000	1.000	0.000	0.769	0.769	0.588	0.667	0.765
	5	0.000	1.000	1.000	0.000	0.462	0.615	0.000	0.606	0.588
	6	0.000	1.000	0.000	0.000	1.000	0.000	0.449	0.449	0.590
	7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
ERP	1	0.000	0.000	0.400	0.261	0.655	0.744	0.000	0.182	0.583
	2	0.000	0.133	0.204	0.210	0.210	0.886	0.000	0.182	0.167
	3	0.267	0.400	0.526	1.000	1.000	1.000	0.250	0.250	0.417
	4	0.000	0.000	0.000	0.089	0.267	0.838	0.000	0.182	0.472
	5	0.000	0.000	0.400	0.364	0.364	1.000	0.000	0.182	0.306
	6	0.000	0.000	0.278	0.000	0.000	0.140	0.000	0.000	0.000
	7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
BPI2016	1	0.067	1.000	0.067	0.000	0.000	1.000	0.005	0.671	0.260
	2	0.067	1.000	0.067	0.000	0.000	1.000	0.000	0.671	0.038
	3	0.000	1.000	0.233	0.000	0.421	0.011	0.000	0.671	0.043
	4	0.067	1.000	0.067	0.000	0.000	1.000	0.000	0.671	0.038
	5	0.045	1.000	0.122	0.211	0.211	1.000	0.000	0.671	0.029
	6	0.022	0.799	0.038	0.000	0.000	1.000	$-\infty$	$-\infty$	$-\infty$
	7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
UWV	1	1.000	1.000	1.000	0.333	1.000	1.000	0.435	0.959	0.913
	2	1.000	1.000	1.000	0.333	1.000	1.000	0.435	0.959	0.913
	3	0.000	0.259	0.187	0.000	1.000	0.879	0.000	0.959	0.877
	4	1.000	1.000	1.000	0.333	1.000	1.000	0.435	0.959	0.913
	5	1.000	1.000	1.000	0.331	0.943	0.510	0.435	0.959	0.913
	6	1.000	1.000	1.000	1.000	1.000	1.000	0.920	0.959	0.935
	7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

to add this in any prompt, nor did we include it in our gold standard, which as a logical choice simply uses table names as ActivityIDs when no dedicated ActivityID column is available.

The mismatch in the ActivityIDs between the generated event log and the gold standard may also explain why the PMC_2 and PMC_3 scores are better for all prompts. PMC_2 ignores ActivityIDs, while PMC_3 does not require that ActivityIDs match 100% to be considered a true positive. Figure 2b shows that when the ActivityID is ignored, the F1-scores are all above zero.

Another pattern that can be observed is the relation between database complexity (V3 being more complex than V2, which is more complex than V1) and the F1-score, with more complex databases reaching lower scores. This implies that next to a mismatch on ActivityID other mismatches are present. Further



(a) PMC_1 (b) PMC_2 (c) PMC_3

Fig. 2: P2P F1-scores for the different used prompts and PMCs.

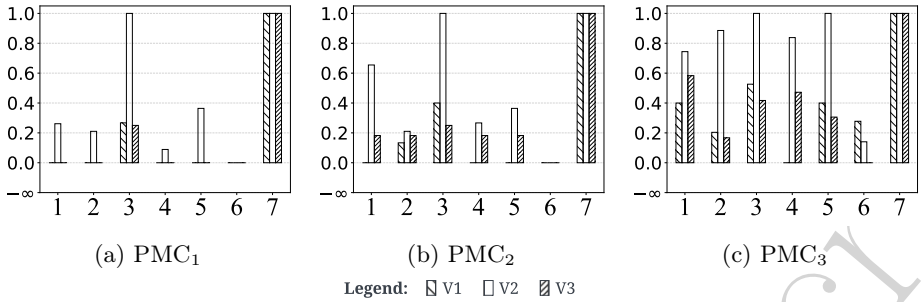
examining the generated event logs reveals that either the case notion is not correctly derived (e.g., some events have InvoiceID as the CaseID instead of OrderID), or the generated event log incorporates irrelevant events. In other words, events from tables that are present in the DB schema but do not have a relation with an order (e.g., click events from a customer).

Mismatches in case notion and inclusion of irrelevant events also affect the PMC_3 scores, shown in Figure 2c. Most PMC_3 F1-scores are equal or higher than the PMC_2 F1-scores. Apparently the ActivityIDs are close enough to the gold standard ActivityIDs to not negatively impact the F1-score. This means that the match between the ActivityIDs of the generated event logs are above the threshold of 0.75. This is not the case for the *process mining knowledge* prompt in combination with the databases V1 and V2. None of the ActivityIDs of the generated event log come close enough to the gold standard which results in an F1-score of zero. This is caused by the prompt itself, which requests the LLM to create an ActivityID by concatenating the table name and the column name in the situation that no other columns exist that could be the ActivityID. However, this should only be done when at least two timestamp columns are present in a table. Otherwise the name of the table should be used as the ActivityID. In this case we only have one timestamp column, hence the LLM applied the wrong part of the prompt.

4.2 ERP Database

The ERP database has a similar complexity to the P2P database, as can be observed from the characteristics of both databases in Table 2. The biggest differences are the additional tables, the many-to-many foreign key relationships among the tables, and an extra case notion. The additional tables represent order lines, payment lines, and shipment lines, which are respectively related to orders, payments, and shipments. These additional tables are needed to connect, for example, payments to invoices. In the P2P database this information was part of the payment table.

The F1-scores for the three versions of the ERP database are shown in Figure 3. The scores of PMC_1 are similar to those of the P2P database, with



(a) PMC_1 (b) PMC_2 (c) PMC_3

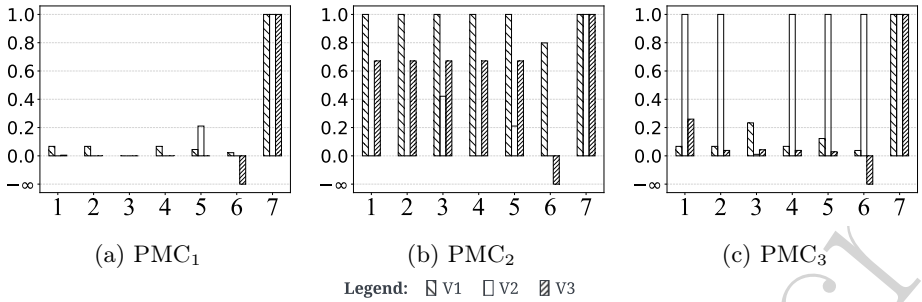
Fig. 3: ERP F1-scores for the different used prompts and PMCs.

prompt 7 performing perfectly, but the other ones performing poorly with many F1-scores at zero. V2 of the database is performing slightly better, with a perfect F1-score for prompt 3 and 7 but scoring below 0.4 for the other ones. We see similar results for PMC_2, which performs slightly better, and PMC_3, which performs best in terms of the ERP database. However, both perform worse than in the P2P database. Overall, the F1-scores for V2 are generally better than for V1 and V3. Prompt 7 consistently reaches F1-scores of 1, while the performance of the other prompts varies greatly.

From the results of PMC_2 (cf., Figure 3b), we gather that even ignoring the ActivityID is not enough to generate an event log close to the gold standard. Only the *custom-made* prompt performs perfectly, as it was designed to do. This indicates that it is possible to write an effective prompt to support the event log extraction for the ERP database. Moreover, the ERP database V2 stands out in all the F1-scores in Figure 3. Considering that V2 is a less complex and V3 is the most complex database, we again notice that there is a relation between the complexity of the database and the F1-scores, as F1-scores for less complex databases are generally higher. In addition, we notice that the prompt type also matters, as different prompts yield different F1-scores for the ERP database. Prompt type 3 (*Few-shot example*) relatively performs best compared to the other prompt engineering strategies.

4.3 BPI2016 Database

The complexity in the BPI2016 databases comes from having multiple columns that can serve as ActivityID and multiple timestamp columns per table, giving the LLM more options to choose from. Figure 4 shows the different PMCs for the three versions of the BPI2016 database. The F1-scores for PMC_1 are the lowest of all the four databases, with all database versions performing poorly except prompt 7, which again scores perfectly. There is even one F1-score that could not be calculated, namely prompt 6, which is depicted in the figure as a negative value. The performance for PMC_2 varies greatly, with V1 scoring perfectly on all prompts except prompt 6, where it scores 0.8. V3 scores around 0.7 for prompts 1 through 5 but again scores negatively for prompt 6. V2 performs worst, with



(a) PMC_1 (b) PMC_2 (c) PMC_3

Fig. 4: BPI2016 F1-scores for the different used prompts and PMCs.

many F1-scores at zero. This is different for PMC_3, where V2 performs well, with F1-scores of 1 for all prompts except 3. Prompt 7 again scores perfectly, but all other prompts perform badly for V1 and V3, all scoring lower than 0.3.

As for the F1-scores that could not be calculated: the reason is that the LLM did not return a valid SQL statement. Without a valid SQL statement an event log cannot be generated. The generated SQL statement included a join of two tables that both contained the column *page_action_detail*. This column was selected as the ActivityID. Unfortunately, the SQL SELECT statement did not specify from which table this column should be taken. This resulted in an *ambiguous column* error.

Figures 4b and 4c show the influence of the ActivityID on the calculated F1-scores. The values of the PMC_2 (cf., Figure 4b) are at least 0.6, except for the database V2. This implies that for the V1 and the V3 databases when the ActivityID is left out of the match, the generated event log resembles the gold standard event log. The inverse of Figure 4b is visible in Figure 4c. In this case, database V2 has high F1-scores for nearly every prompt, while V1 and V3 have low scores. This effect is caused by the composition of the BPI2016 database versions. V1 contains questions, messages and complaints. V2 contains website clicks for customers who are known because they logged in and clicks from unknown customers. V1 and V2 have disjunct sets of tables. In V3 all the tables from V1 and V2 are combined.

Figure 4b also shows that the LLM is capable of retrieving the correct CaseID and timestamps for V1 and V3. Looking into the generated SQL statement for V2, it is revealed that the LLM choose the wrong CaseID, i.e., SessionID instead of CustomerID. In the PMC_3 F1-scores in Figure 4c the ActivityID plays a role once more. Even though the LLM selected the wrong CaseID for V2, the match is considered perfect, judging by the F1 scores of 1 for almost every prompt. This turns out to be an artifact from the Levenshtein distance comparison that is used in our PMC_3. The SessionID for each event is very close to the CustomerID, i.e, both columns contain numbers that appear to be very similar. The match result is just above the threshold for recording a TP. If the threshold would be lowered, then Figure 4c would resemble Figure 4a. Finally, the PMC_3 F1-scores for V1 and V3 in Figure 4c are low because the LLM selected the

wrong column as ActivityID. The values of this column are, even with PMC_3, not similar enough to the ActivityIDs in the gold standard.

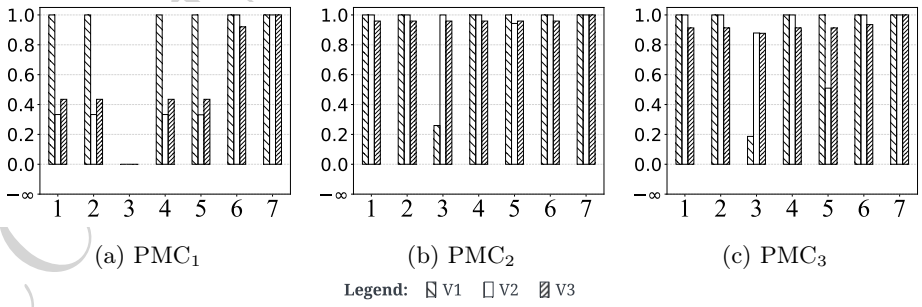
In conclusion, the LLM does not seem to handle the BPI2016 database well. Either the wrong CaseID is selected (for V2) or the wrong ActivityID is selected (for V1 and V3). For the timestamp, the correct column is selected by the LLM. The complexity of this database mostly comes from having multiple columns that can serve as ActivityID.

4.4 UWV Database

The UWV database is from a typical claim handling process as executed by UWV. It combines tables from different information systems used in the execution of the process. The complexity of this database lies in the lack of direct connections between the tables, as there are no tables containing foreign keys. Next to that, there is one table containing multiple timestamp columns. Each of these columns represents a different event.

Figure 5 contains all PMC F1-scores for the UWV databases. Most of the F1-scores have the maximum possible value of 1. V1 especially, performs well, scoring 1 for all prompts across PMCs, except for prompt 3 where it scores below 0.3 for all three PMCs. V2 and V3 perform very well for PMC_2 and PMC_3, mostly nearing the maximum scores, but experience more difficulty with PMC_1, where they score below 0.5 for prompts 1 through 5.

For all three PMCs, the LLM is capable of generating an event log that is in line with the gold standard. In addition to the *custom-made* prompt (prompt 7), the *process mining knowledge* prompt (prompt 6) also performs well. For the UWV database V1, most prompts work well, except for the *few-shot example* prompt (prompt 3), which generates incorrect SQL statements referencing orders and payments not present in the database. This only occurs for the UWV database V1. For the other database versions the *few-shot example* prompt (prompt 3) is interpreted as an example, as it should be.



(a) PMC_1 (b) PMC_2 (c) PMC_3

Fig. 5: UWV F1-scores for the different used prompts and PMCs.

5 Discussion

After executing a variety of prompt strategies against a number of databases with different structural characteristics, we have acquired two major insights. First, we learned that prompt engineering strategies per se are not enough for effective LLM-assisted event log extraction. Our results show that LLMs cannot fully autonomously identify effectively what the desired event log should look like, nor what case notion should be considered. Second, we learned that domain and data knowledge can make LLM-assisted event log extraction effective. What is interesting is that having domain and data knowledge, and being able to transfer those to the LLM, can yield effective LLM-assisted event log extraction. Particularly, if there is previous knowledge available about which specific tables hold which event data, or if the LLM is provided with a couple of examples of how the event log extraction should be performed, the outcome of the LLM-assisted event log extraction becomes more effective.

Based on these findings, we can make three specific recommendations to support the effective use of LLM-assisted event log extraction:

R1 - Combine prompt engineering strategies with domain and data knowledge. Prompt engineering strategies such as *persona*, *few-shot examples*, *tree-of-thought* or *chain-of-thought* should be combined with domain and data knowledge to produce more effective results. Considering only the *custom-made* prompts, which leverages domain and data knowledge, LLM-assisted event log extraction could produce useful event logs in 100% of the times (36 out of 36 runs). In the context of our work, an event log is considered useful if its F1-score is above 0.75 for either PMC_1, PMC_2 or PMC_3.

R2 - Do not rely on the first generated event log. If domain and data knowledge are not readily available, be ready to test the generated event log and go back to the LLM for refinement of the event log extraction. Considering our results for all prompts (and not only for the *custom-made* prompt), LLM-assisted event log extraction can produce useful event logs in 68.3% of the times (172 of the 252 runs).

R3 - Use LLM-assisted event log extraction with parsimony. If you do not know how to extract event logs and you need to start analyzing some particular process data, you can leverage LLM-assisted event log extraction considering the recommendations R1 and R2. However, for more stable and precise process analysis, double-check the event log extraction with a domain or data expert.

6 Conclusion

In this paper, we presented a first experimental setup to investigate the effectiveness of LLMs in assisting event log extraction from a relational database. We showed that there is no one-fits-all prompt engineering strategy solution for databases with different structural characteristics and that domain and data knowledge is still needed to yield satisfactory results. We also showed that if

domain and data knowledge is available, one can actually benefit from an LLM-assisted event log extraction, as it can create and adapt SQL scripts fast, enabling business analysts to get to the process discovery and analysis step for process mining faster. Our findings, therefore, show the potential for LLM-assisted event log extraction. They pave the way for future *human-as-a-tool* in a human-LLM interaction event log extraction system, where the user is actively asked by the LLM for clarifications about domain and data related ambiguities, to diminish the error-proneness inherent in event log extraction.

Naturally, our work is not without limitations. While our analysis involved multiple databases to enhance robustness of our experiment, the relatively small database sample size may impact the generalizability of our findings. Also, because of the limitations imposed by the prompt length. We, however, used a diverse range of prompts and databases to ensure a systematic selection process to cover varied scenarios. Moreover, we established clear data collection and analysis protocols, which ensured consistency and reduced individual bias in the interpretation of data. While we do not claim to provide a complete set of test cases, with this work we provide an architectural basis for future research to systematically assess other combinations of databases, prompts, performance metric calculations, and large language models in assisting event log extraction for process mining.

In future work, we aim at incorporating our findings into a multi-agent framework for event log extraction, where data from different sources can be jointly considered, and an LLM-assisted generation of object-centric event logs can also be investigated.

Acknowledgements. Part of this research was funded by NWO (Netherlands Organisation for Scientific Research) project number 16672.

References

1. van der Aa, H., Leopold, H., Reijers, H.A.: Comparing textual descriptions to process models – the automatic detection of inconsistencies. Information Systems **64**, 447–460 (2017)
2. van der Aalst, W.M.P.: Process mining: Discovery, conformance and enhancement of business processes. Springer (2011)
3. 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. Springer (2024)
4. Brown, T., et al.: Language models are few-shot learners. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 33. Curran Associates (2020)
5. Busch, K., Rochlitzer, A., Sola, D., Leopold, H.: Just tell me: Prompt engineering in business process management. In: BPMDS. Springer (2023)
6. Dees, M., van Dongen, B.F.: BPI Challenge 2016 (2016)
7. Dijkman, R., Dumas, M., Dongen, van, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems **36** (2011)
8. Finlayson, M., Ren, X., Swayamdipta, S.: Logits of API-Protected LLMs leak proprietary information. ArXiv **2403.09539** (2024)

9. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-SQL empowered by large language models: A benchmark evaluation. Proceedings of the VLDB Endowment **17**(5), 1132–1145 (2024)
10. Jiang, A.Q., et al.: Mistral 7B. ArXiv **2310.06825** (2023)
11. Kasneci, E., et al.: ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences **103** (2023)
12. Kourani, H., Berti, A., Schuster, D., van der Aalst, W.M.P.: Process modeling with large language models. In: Enterprise, Business-Process and Information Systems Modeling. Springer (2024)
13. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics-Doklady **10**(8) (1966)
14. Li, C.Y., Joshi, A., Tam, N.T.L., Lau, S.S.F., Huang, J., Shinde, T., van der Aalst, W.M.P.: Rectify sensor data in IoT: A case study on enabling process mining for logistic process in an air cargo terminal. In: Cooperative Information Systems (CoopIS). pp. 293–310. Springer (2024)
15. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: Information Systems in the Big Data Era. Springer (2018)
16. Mustroph, H., Winter, K., Rinderle-Ma, S.: Social network mining from natural language text and event logs for compliance deviation detection. In: Cooperative Information Systems (CoopIS). pp. 347–365. Springer (2024)
17. Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Cooper, A.F., Ippolito, D., Choquette-Choo, C.A., Wallace, E., Tramèr, F., Lee, K.: Scalable extraction of training data from (production) language models. ArXiv **2311.17035** (2023)
18. Ollion, É., Shen, R., Macanovic, A., Chatelain, A.: The dangers of using proprietary LLMs for research. Nature Machine Intelligence **6**(1), 4–5 (2024)
19. Stein Dani, V., Leopold, H., van der Werf, J.M.E.M., Lu, X., Beerepoot, I., Koorn, J.J., Reijers, H.A.: Towards understanding the role of the human in event log extraction. In: Business Process Management Workshops. Springer (2022)
20. Teubner, T., Flath, C.M., Weinhardt, C., van der Aalst, W., Hinz, O.: Welcome to the era of ChatGPT et al. Business & Information Systems Engineering **65**(2), 95–101 (2023)
21. Thirunavukarasu, A.J., Ting, D.S.J., Elangovan, K., Gutierrez, L., Tan, T.F., Ting, D.S.W.: Large language models in medicine. Nature Medicine pp. 1930–1940 (2023)
22. Touvron, H., et al.: Llama 2: Open foundation and fine-tuned chat models. ArXiv **2307.09288** (2023)
23. Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 35, pp. 24824–24837. Curran Associates (2022)
24. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D.C.: A prompt pattern catalog to enhance prompt engineering with ChatGPT. ArXiv **2302.11382** (2023)
25. Wu, Y.: Large language model and text generation, pp. 265–297. Springer (2024)
26. Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Zhong, S., Yin, B., Hu, X.: Harnessing the power of LLMs in practice: A survey on ChatGPT and beyond. ACM Transactions on Knowledge Discovery from Data **18**(6) (2024)
27. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models. In: Advances in Neural Information Processing Systems (NeurIPS). Curran Associates (2023)