Redesigning Business Processes to Reduce Waiting Times Using Large Language Models

Katsiaryna Lashkevich*, Fredrik Milani, Maksym Avramenko* and Marlon Dumas

University of Tartu, 18 Narva mnt, Tartu, 51009, Estonia

Abstract

Reducing waiting times in business processes is crucial for minimizing waste and improving efficiency. To achieve this, analysts need to understand the causes of waiting times and identify relevant redesign options. To support this task, we introduce a conversational interface integrated with the Kronos tool to analyze waiting time causes and suggest process redesigns. The interface uses a Large Language Model (LLM) to query waiting time analysis results and contextual process data based on specific user requests. The tool additionally provides recommendations of redesign options to reduce the identified waiting times, based on a collection of redesign heuristics.

Keywords

process mining, waiting time, large language models

1. Introduction

Understanding where and why waiting times occur in business processes is essential for reducing waiting time-related wastes and improving process efficiency. Waiting times occur when a case transitions from one activity to another [1]. For instance, in a procurement process, waiting time may occur when a purchase order is approved and the order is waiting to be fulfilled or between receipt of goods and their inspection for approval. Identifying why such waiting times occur helps analysts to pinpoint inefficiencies and to assess possible redesign options.

Process mining techniques support the discovery of waiting times from event logs that record business process executions [2]. Existing process mining tools [3], can identify and analyze waiting times from event logs. For example, Kronos [4] – an open-source web-based tool – can both analyze waiting times and identify their causes. However, existing tools have two limitations. First, the results are typically presented through visualizations with basic interactivity, such as filtering and hiding data, but lack the capability for exploratory interaction to support customized analyst requests. Second, while these tools effectively analyze waiting times and their causes, they do not offer redesign suggestions for reducing waiting times.

Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 22nd International Conference on Business Process Management (BPM 2024), Krakow, Poland, September 1st to 6th, 2024.

*Corresponding author.

katsiaryna.lashkevich@ut.ee (K. Lashkevich); fredrik.milani@ut.ee (F. Milani); maksym.avramenko@ut.ee (M. Avramenko); marlon.dumas@ut.ee (M. Dumas)

0000-0003-4426-7738 (K. Lashkevich); 0000-0002-1322-915X (F. Milani); 0009-0004-0725-7623 (M. Avramenko); 0000-0002-9247-7476 (M. Dumas)

In this paper, we introduce a tool integrated with Kronos [4] that leverages a Large Language Model (LLM) to enable users to conversationally query event log data to analyze the causes of waiting times in a business process, and to obtain redesign suggestions for waiting time reduction. The tool is designed to discover and to quantify waiting times in a business process attributable to batching, prioritization, resource contention, and resource unavailability. Users interact with the tool via a chatbot interface. For each identified cause of waiting time, the tool provides redesign suggestions based on a catalog of redesign strategies.

2. Tool Overview

2.1. Architecture

The tool is based on an existing process mining tool called Kronos and a set of additional algorithms that, collectively, discover and analyze causes of waiting times from event logs and provide redesign options. Figure 1 illustrates the architecture of the tool, highlighting the main components of the conversational interface.

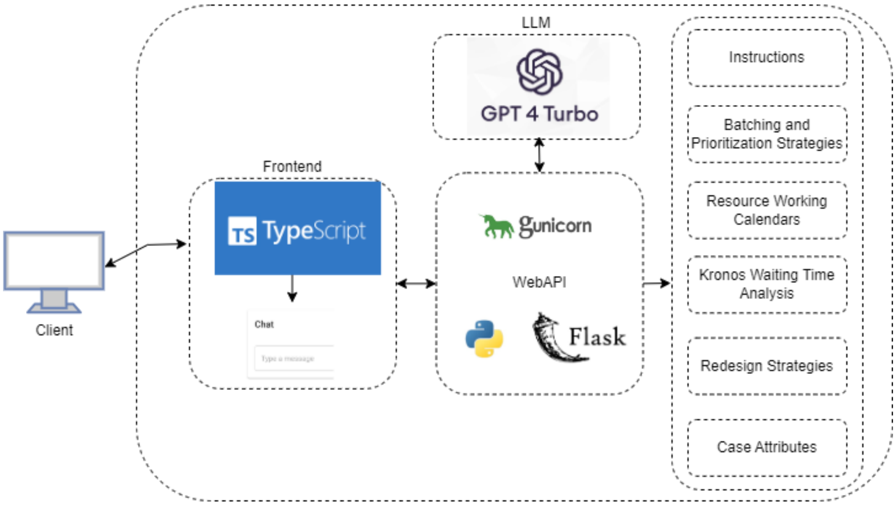

Figure 1: Tool architecture

The interface has the following components. The frontend is developed using TypeScript and provides a simple chat interface for users to submit their queries. The user inputs their question, which is then forwarded to the backend for processing. The WebAPI, implemented using Flask and Gunicorn, acts as the intermediary between the frontend and the LLM. It handles the query forwarding, data retrieval, and response generation processes. The tool uses OpenAI's GPT-4 Turbo model for its language processing capabilities.

The LLM accesses several data sources to provide relevant responses, including waiting time analysis from Kronos, batching and prioritization strategies, resource working calendars, case attributes, and redesign strategies. We use OpenAI's "function calling"[1] to set up these data

[1] https://platform.openai.com/docs/guides/function-calling

sources available for the LLM.

- **Kronos Waiting Time Analysis:** Kronos is the primary source of information, providing a detailed analysis of waiting time causes. The analysis results are stored in a database. Instructions were provided to the LLM to make it aware of the database schema. This enables the LLM to generate SQL queries, which are executed by the interface on the database. The retrieved information is sent back to the LLM for further processing.
- **Batching Strategies:** Given an event log, batching strategies are discovered using an existing algorithm [5] that analyzes how activity instances are grouped into batches and provides insights into the batch size, frequency, and activation rules[2].
- **Prioritization Strategies** are identified using another algorithm [5], which examines how tasks are prioritized in the process. This analysis helps in understanding the order in which tasks are performed and their impact on waiting times[3].
- **Resource Working Calendars** detail the availability schedules of resources. They are discovered using an existing algorithm [6] that provides information on how resource availability affects waiting times[4].
- **Case Attributes** are discovered from the event log by the tool. These attributes provide additional context about the cases, such as their characteristics and the specific conditions under which they are processed.
- **Redesign Strategies:** A compiled file containing different redesign strategies, their applications, examples, and references. The LLM is provided with a structured list of possible redesign strategies, enabling it to request specific information (description, example, references) on these strategies as needed.

The LLM is provided with pre-configured instructions to enhance its capabilities in interpreting queries and accessing/reading the data sources. Details of the prompts that the tool provides to the LLM, including the collection of redesign strategies, can be found in [7].

2.2. User Flow

Figure 2 depicts the steps performed by the tool to respond to user queries. The process begins with a user submitting a question (step ‘A’), which is forwarded to the LLM (step ‘B’). The LLM performs a data requirement analysis (step ‘C’) to determine what specific data is required to generate a response. Next, relevant data is retrieved (step ‘D’), sent back to the LLM (step ‘E’), which generates a response and presents the final output to the user (step ‘F’).

The interface has a text input box where users can type their questions (see Figure 3). Once submitted, the question appears in the chat box, and the “Send” button is disabled to prevent more questions until a response is provided. The interface dynamically adjusts its size based on the length of the questions and answers, ensuring that all content is readable.

[2] https://github.com/AutomatedProcessImprovement/pix-framework/blob/main/src/pix_framework/discovery/batch_processing

[3] https://github.com/AutomatedProcessImprovement/pix-framework/tree/main/src/pix_framework/discovery/prioritization

[4] https://github.com/AutomatedProcessImprovement/pix-framework/tree/main/src/pix_framework/discovery/resource_calendar_and_performance

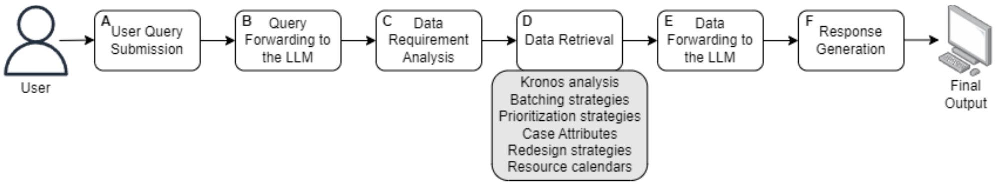

Figure 2: Overview of the conversational interface workflow.

Figure 3: Chat interface for user question submission.

Upon the first question submission (step 'A'), a thread is created[5] and an ID is assigned, which is kept in the session storage. Subsequent questions will use the same thread ID, allowing the LLM to be aware of the history of previous messages and answers, as well as the data connected to those interactions. If the user refreshes the page, the thread ID is no longer retained in memory, prompting the tool to create a new thread upon the next question submission. This ensures that each session maintains a dialogue history unless manually reset by the user.

Once a question is submitted (step 'A'), it is forwarded to the OpenAI GPT-4 Turbo LLM (step 'B') via their API[6]. To ensure the LLM understands its operational context and meets the requirements for analyzing event logs and suggesting process improvements, we have provided it with structured instructions[7]. The instructions are tiered at three levels to enhance the LLM's responsiveness. The first is *LLM's Instructions*, which contains general guidelines such as the LLM's goals and operational context. The second is *General Message Instructions*, specifying preferred data formats, instructions on analyzing waiting time causes and recommending the redesign options. The last is *Direct Message Instructions* which refine responses by concatenating them with user queries, ensuring, e.g., that responses follow a consistent time format. General and direct message instructions are included in each user message.

Based on the data request from the LLM, the necessary data is retrieved and provided to the LLM (step 'E'), enabling it to proceed with the final response generation (step 'F'). If an error occurs during the data retrieval process, an error message is returned to the LLM, allowing it to adjust the data request accordingly. Having analyzed the initial question from the user, the available data sources, and the knowledge of the LLM, the LLM generates the final response (step 'F'). This response is then presented to the user. The LLM uses markdown, supported by the interface, to secure that the user receives the answer in a structured and readable format. After the response is generated, the user can ask additional questions, with the LLM taking into account the context of previous messages.

[5] https://platform.openai.com/docs/api-reference/threads/object
[6] https://openai.com/index/openai-api/
[7] https://github.com/AutomatedProcessImprovement/waiting-time-backend/blob/main/assistant-instructions.txt

3. Maturity and Availability

The conversational LLM interface is a proof-of-concept. An initial user validation has been conducted with process mining experts. The preliminary results indicate the tool's usefulness for conversational exploration of waiting time causes and process redesigns.

The conversational interface is publicly available at http://kronos.cloud.ut.ee/. To use the interface, users must first upload an event log to Kronos. The implementation of the interface logic is available in GitHub repositories: one for the frontend[8] and one for the WebAPI[9], together with instructions on usage. An API key from OpenAI is needed to start the tool and generate answers. A screencast that describes the tool is available on YouTube[10].

The primary limitation of our tool is its reliance on pre-configured LLM instructions, which may not cover all user queries. The implementation is also constrained by the LLM performance and the efficiency of data retrieval processes. Response times can affect user experience, and result accuracy depends on event log data quality and predefined queries.

4. Conclusion

The conversational interface integrated with Kronos helps users explore waiting time causes and obtain redesign suggestions. Using OpenAI's GPT-4 Turbo model, it provides dynamic responses to queries about waiting time causes and process redesigns. Structured instructions and function calling enable data retrieval from multiple sources to generate relevant responses. The interface supports iterative question-answering, maintaining context across sessions.

Acknowledgments Work funded by the European Research Council (PIX project).

References

[1] K. Lashkevich, F. Milani, D. Chapela-Campa, I. Suvorau, M. Dumas, Why am I waiting? Data-driven analysis of waiting times in business processes, in: CAiSE, Springer, 2023.

[2] W. M. P. van der Aalst, Process Mining: Data Science in Action, 2nd ed., Springer, 2016.

[3] M. A. Ali, F. Milani, M. Dumas, Data-driven identification and analysis of waiting times in business processes: A systematic literature review, Bus. Inf. Syst. Eng. (2024). In press, https://doi.org/10.1007/s12599-024-00868-5.

[4] K. Lashkevich, F. Milani, D. Chapela-Campa, I. Suvorau, M. Dumas, Kronos: Discovery and analysis of waiting time causes., in: ICPM Doctoral Consortium/Demo, 2023.

[5] I. Halenok, Business process simulation with differentiated resources, 2023.

[6] O. López-Pintado, M. Dumas, Business process simulation with differentiated resources: Does it make a difference?, in: BPM, Springer, 2022.

[7] K. Lashkevich, F. Milani, M. Avramenko, M. Dumas, LLM-assisted optimization of waiting time in business processes: A prompting approach, in: BPM 2024, Springer, 2024. To appear, preprint available at: https://owncloud.ut.ee/owncloud/s/zy66MaTxQpK7wbk.

[8] https://github.com/AutomatedProcessImprovement/waiting-time-frontend

[9] https://github.com/AutomatedProcessImprovement/waiting-time-backend

[10] https://youtu.be/ScxjMnjqPc8