This document is part of the supplementary material for the paper titled 'LLM-Enabled Business Process Coherence Checking Based on Multi-Level Process Documentation' by Schulte, M.; Franzoi*, S.; Köhne, F.; vom Brocke, J. submitted for publication to the journal Process Science (https://link.springer.com/journal/44311).

Section D:

Technology Stack

The technology stack used for aProCheCk is critical to its functionality and performance. Each component has been carefully selected to ensure compatibility, efficiency and compliance with relevant standards.

The code for this artifact is written in Python. Python's prominence in data science and LLMs makes it the ideal choice due to its extensive library support and compatibility with various tools and frameworks (Tu et al., 2023). This choice ensures that our solution is built on a solid and widely adopted foundation. For development purposes, Visual Studio (VS) Code was used as the integrated development environment (IDE), providing an efficient and robust platform for coding. To work with BPMN models, the BPMN.io Editor plugin for VS Code was used. This plugin is based on the free web-based BPMN.io tool from Camunda[3], which can be used to create and edit BPMN diagrams in a browser application. This tool was instrumental in creating and managing BPMN models such as the dataset discussed in Section C.

Azure OpenAI GPT-4o was selected for the LLM. This model was selected based on its leading performance, as demonstrated by benchmarks in business process related tasks, as well as its availability and reliability (Berti et al., 2024). Importantly, it can be hosted within the European Union and thus complies with the European General Data Protection Regulation (GDPR) standards. This compliance is essential for working with sensitive corporate data, as indicated in the interviews with practitioners, and is particularly relevant for demonstrations in the focus groups, which are based on naturalistic data. Furthermore, because Azure OpenAI GPT-4o is accessed via a REST API, the LLM component remains modular and can be replaced with another model or service without affecting the core functionality of the system.

The process documentation database in the prototype is represented by a simple folder structure, as opposed to, for example, a relational database. Each process has its own folder containing all associated process documents. This simplified approach allows for easy access and management of documents during the development phase. In a production environment, this could be replaced by an interface to the database of a process modelling software or a corporate knowledge base, as suggested during the expert interviews.

In summary, the chosen technology stack ensures that aProCheCk is built on a robust, flexible and secure foundation that can evolve to meet future needs and integrate with existing systems.

Prototype Development

Prototype development is the practical implementation of the artifact's conceptual design, characterised by continuous iteration and refinement. This phase translates theoretical insights and refined design specifications into a functional prototype, ensuring that it meets both academic and practical requirements.

The development process in a DSR project is inherently iterative (Hevner et al.,

2024). The development of aProCheCk focuses on modular architecture, structured data processing and a high adaptability. Each development cycle integrates feedback and test results to iteratively improve the functionality and reliability of the prototype. This iterative approach ensures that the prototype evolves to effectively address the challenges of coherence checking in multi-level process documentation.

This section examines the core components of the prototype. First, the overall software architecture, designed for flexibility, modularity and scalability, is presented to demonstrate how seamless integration of different components is achieved. This is followed by an exploration of data preprocessing techniques. These are used to prepare process documentation for coherence analysis. Finally, the interaction mechanisms that underpin the prototype are detailed, showing the role of structured prompts and efficient context management in achieving accurate and actionable results. The focus throughout these subsections is to illustrate how each component contributes to the iterative refinement and practical effectiveness of the prototype.

Software Architecture. The software architecture of aProCheCk is designed with a high degree of modularity to support flexibility and adaptability. The overall workflow of the system consists of several key components. The software architecture is visualized in Figure 6. The elements in dark grey highlight the architectural components that orchestrate the main logic and workflow of the artifact. The white elements represent the supportive modules, which are associated with particular subtasks. The light grey elements are external components, such as the database and LLM API. The elements with dotted borders highlight optional components that vary for each instantiation of aProCheCk. The instantiations of aProCheCk describe the incorporation of the tool into different environments with different interfaces. In the context of this paper, two instantiations are created. The first instantiation is for the experiments, including experimental evaluations, which is shown in Figure 1 below.

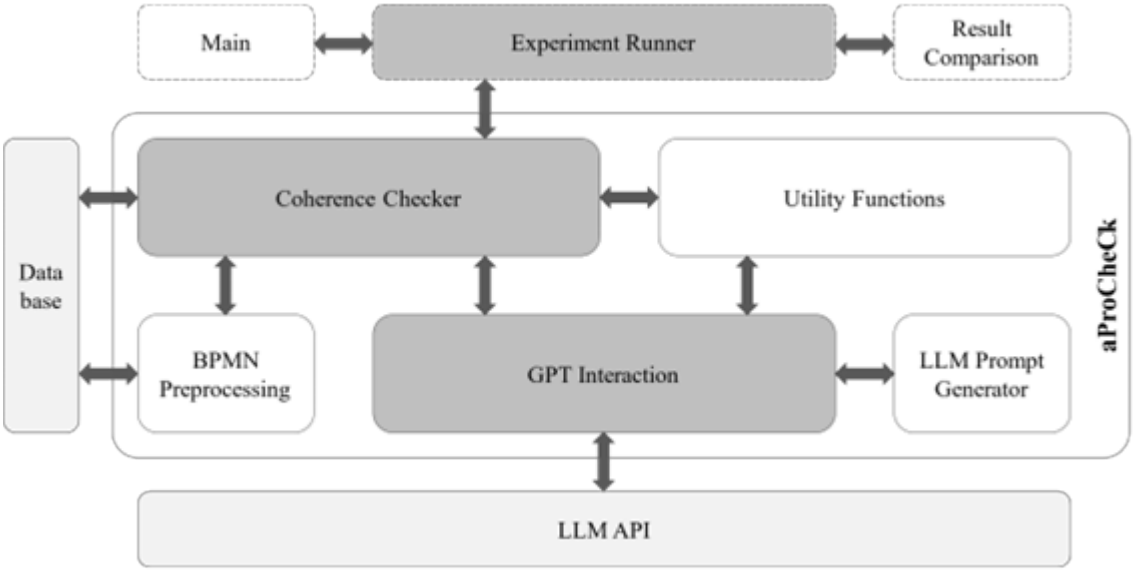

Fig. 1 aProCheCk Software Architecture Visualization.

The aProCheCk artifact is comprised of five core modules, each of which is responsible for a distinct function within the system. The *Coherence Checker* module orchestrates the overall coherence checking process, coordinating with other components and ensuring comprehensive analysis of process documentation. It calls upon the *Utility Functions* module for basic operations such as loading configurations, managing files and converting relevant data types. The *BPMN Preprocessing* module prepares BPMN files by removing non-essential visual elements and ensuring they are in an optimal format for coherence analysis.

The interaction with the LLM is managed by the *GPT Interaction* module, which handles interactions with the external LLM API. This module is responsible for processing requests and managing responses from the LLM. To enable efficient LLM interactions, the *LLM Prompt Generator* module stores predefined templates for LLM prompts, ensuring a standardised structure for task requests sent to the LLM.

Additionally, the architectural design incorporates external, modular components, including a *Database* module, which can be integrated and adapted seamlessly as required. The database, which is necessary for the storage of process documentation, is accessed by both the Coherence Checker and the BPMN Preprocessing modules. Similarly, the *LLM API* is an external component accessed by the GPT Interaction module, thereby facilitating flexible integration and potential updates or replacements with different LLM services without disrupting the artifact's core functionality.

In order to conduct an iterative experiment benchmarking process, three additional modules have been incorporated into the instantiation of aProCheCk. In particular, these modules are the Main, Experiment Runner, and Result Comparison modules. Although these modules are not part of the main software architecture, they are required for the execution and evaluation of the experiments. The *Main* module initiates the experiments, while the *Experiment Runner* module handles the implementation of the experimental processes, including the execution of multiple iterations of coherence checks. The *Result Comparison* module is responsible for analysing the experimental outcomes and comparing them against the expected results, thus facilitating a comprehensive assessment of the artifact's performance.

The prototype is designed without a graphical user interface (GUI), as this is not required for the evaluation purposes of this paper and the desired operational model of the artifact. Instead, the output is provided in a structured JSON format, which allows for seamless integration into any existing user interface. For the instantiation, the generation of notifications in the structure of emails was implemented using an additional LLM API call based on the structured JSON data.

In summary, the software architecture of aProCheCk emphasizes modularity and flexibility, ensuring that the system can be adapted to existing technologies and desired LLM services while maintaining robust and efficient performance.

Data Preprocessing. The preprocessing process begins with the identification of relevant files, which is tailored to each instantiation of aProCheCk. In particular, the instantiation used

for the experiments is informed by insights gained from expert interviews, using timestamps of the most recent modification as the primary criterion. The identification process is based on the specified database and the processes for which the artifact is triggered to check coherence.

The *Utility Functions* module performs the file identification process by retrieving the relevant data based on configured criteria, including file types, modification timestamps, and file names. To illustrate, during the experiments, the two most recently modified BPMN files are selected. Of these, the most recently modified file is considered the ground truth, while the other serves as a point of comparison. Additionally, the most recently modified TXT file is identified as the relevant textual document. This selection approach, supported by the expert interviews, ensures that the most recent and relevant data is used for the analysis.

Once the relevant files have been identified, the *BPMN Preprocessing* module automatically preprocesses the BPMN files. This involves removing all elements from the XML representation of the BPMN files that are purely visual, such as coordinates and stylistic details. According to the expert interviews, these visual elements are considered uncritical and do not pose a risk to the relevant process coherence. Essential semantic information, including the organization of elements within pools and their relationships, is retained to preserve the structural integrity of BPMN models. By removing visual BPMN data, the preprocessing step decreases file sizes, reduces noise in the data, and improves the clarity and focus of the coherence analysis, thereby increasing the efficiency and effectiveness of the artifact.

This preprocessing step is integral to the overall process, in alignment with the optimizations described previously. These optimizations are designed to enhance the efficiency and accuracy of the artifact's coherence checks. By focusing on the most relevant data and removing insignificant information, the artifact is better positioned to execute its intended functions effectively.

In summary, careful data preprocessing, guided by expert insight and driven by specific criteria, is essential to maintaining the performance of the artifact.

LLM API Interaction. The Azure OpenAI ChatGPT-4o API is a critical component of aProCheCk, enabling the advanced computational capabilities required for coherence checking in process documentation. Introduced in Section D, this API operates under the REST architecture, which ensures a standardised, reliable and scalable method of communication.

As a REST API, the Azure OpenAI ChatGPT-4o interface[4] is stateless. This implies that each API call is processed independently and no information is retained between calls. This stateless nature simplifies interaction, improves scalability and performance, and reduces complexity. In addition, the RESTful design ensures high security, maintainability and seamless performance by decoupling the client-server communication.

A key feature used in this artifact is the API's JSON mode, which allows structured output to be generated according to a JSON schema that is predefined as part of each prompt.

This capability is essential for automating the processing of results, allowing the artifact to systematically parse and interpret the API responses. By ensuring that the output follows a consistent structure, the artifact can efficiently extract and utilise the necessary information without manual intervention.

Interaction with the LLM involves a two-step process, which is crucial for checking the coherence of multi-level process documentation. The first query compares the two versions of the original document. The subsequent prompt then checks whether the changes identified are still coherent with the related process document. This sequential interaction is similar to the self-prompting technique, where part of the initial response feeds into the subsequent prompt. This layered approach, discussed in more detail in Section D, ensures thorough and context-aware coherence checking.

Efficient context management is critical to optimising API interactions. To minimise noise and computational overhead, only selected information relevant to each prompt is provided to the API, without full previous conversations being provided as context. For the second step, only essential parts of the first response are included, for example, the chain of thought descriptions are omitted, avoiding excessive data transfer and focusing the model's attention on relevant information. This selective approach ensures that only relevant data is used, improving both the efficiency and effectiveness of processing.

Currently, aProCheCk does not take advantage of API features such as fine-tuning, embedding, and multimodality. Fine-tuning could improve response quality but is limited by data constraints. Embedding provides deeper model understanding of organizational processes but is omitted due to scale limitations. Multimodality, which allows processing of diverse data types, is not used to maintain model interchangeability and focus on text-based documents. However, these features are promising areas for future research and development.

Overall, interaction with the Azure OpenAI API is a cornerstone of the artifact, leveraging the strengths of the REST architecture and specific API features to provide efficient, maintainable, and powerful coherence checking.

Prompt Engineering. Prompt engineering represents a vital technique for optimising the utility of LLMs across a range of domains, evident in its use in the context of BPM (Busch et al., 2023). Despite its rising significance, prompt engineering remains a relatively new area of research. Only recently established and validated techniques have begun to emerge in the literature, significantly increasing the potential for the use of LLMs (Sahoo et al., 2024). The comprehensive taxonomy proposed by Schulhoff et al. (2024) classifies 58 general text-only prompting techniques into six clusters: Zero-Shot, Few-Shot, Thought Generation, Ensembling, Self-Criticism, and Decomposition.

Among these approaches, Few-Shot Prompting and Chain of Thought Prompting have proven to be particularly effective and are central to this paper due to their proven effectiveness and suitability for the developed artifact (Schulhoff et al., 2024). Few-shot prompting involves providing the model with a limited number of examples from which to learn and perform specific tasks (Sahoo et al., 2024). This technique is particularly valuable in contexts where extensive training data is not available, allowing the model to generalise

effectively from minimal examples. Chain of Thought prompting, the only technique introduced in the Thought Generation category, encourages the LLM to articulate its reasoning process step by step before delivering a final answer (Sahoo et al., 2024). This encourages responses that are more accurate and logically structured. Both techniques are integrated into the iterative experimental optimization process.

Ensembling and Self-Criticism, while offering potential benefits, were excluded from this paper due to resource constraints. Ensembling aggregates multiple responses to produce a final output, which would incur higher computational costs, while Self-Criticism requires the model to critique its own responses, which presents similar resource challenges (Schulhoff et al., 2024). However, these techniques could be the subject of future research. Decomposition techniques are implicitly incorporated into the structured prompting method, similar to traditional divide and conquer strategies.

In the field of BPM, the discipline of prompt engineering is still in its infancy, as evidenced by a limited number of publications, with only a small subset appearing in high-impact journals (Busch et al., 2023). Nevertheless, the field shows considerable potential for future growth. The potential of few-shot and chain-of-thought prompting to improve output quality in the context of BPM has been demonstrated in studies by Ayad and Alsayoud (2024). Busch et al. (2023) suggests a shift in focus from model fine-tuning to prompt engineering in BPM research, a direction that is consistent with the focus of this paper. While advances in LLM technology, exemplified by GPT-4o's context window exceeding that of 2048 tokens for GPT-3 by more than 60 times, have mitigated some initial concerns proposed by Busch et al. (2023), such as limited context windows, the principles of prompt engineering remain crucial for optimising model performance.

The structured approach to the artifact involves a two-stage process of logic-based coherence checking, as illustrated in the following diagram. The first stage of the process is to compare the content of the modified and original documents. During this stage, individual semantic changes are identified and categorised according to the Business Process Change Dimensions introduced in Section C. The second step, the coherence checking step, consists of comparing the previously identified changes with the content of the related document. This comparison includes the description of any required changes to the related document that are necessary to restore process coherence. Here, the identified changes are classified into change categories. The process can be terminated at any stage if coherence is confirmed. The final step, involves the generation of the notification, which is constructed in the form of a textual notification based on the structured output. In addition, a notification title and severity indicator are added based on the results of the coherence check.

This document is part of the supplementary material for the paper titled 'LLM-Enabled Business Process Coherence Checking Based on Multi-Level Process Documentation' by Schulte, M.; Franzoi*, S.; Köhne, F.; vom Brocke, J. submitted for publication to the journal Process Science (https://link.springer.com/journal/44311).

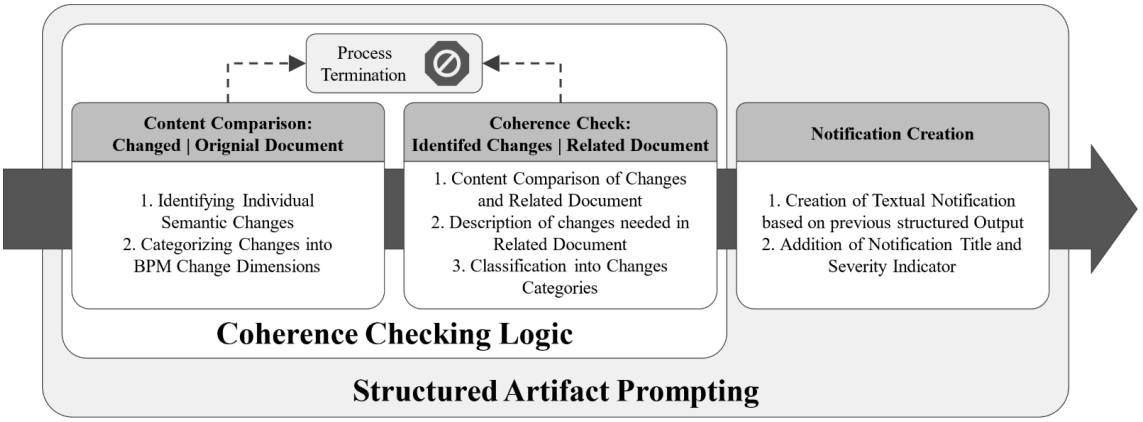

Fig. 2 Structured Prompting Approach for Process Coherence Checking.

Adherence to general prompt engineering best practice is crucial in the development of an LLM-based artifact (Lo, 2023; Marvin et al., 2024). The prompts were structured to place key information at the beginning, ensuring clarity from the outset and facilitating processing by the model. Clear structural divisions within prompts allowed for logical flow and coherence, while the strategic use of keywords highlighted critical points and guided the model's focus. Background information was provided after the task description to provide the necessary context without overwhelming the initial instructions. Step-by-step instructions were used to maintain a logical sequence, with constraints listed at the end to avoid distraction from task performance. Importantly, as part of the experimental iterations introduced in Section E, a system message was set to assign the role of an experienced BPM expert to the LLM, helping to improve its contextual understanding and decision accuracy.

In considering future developments, frameworks such as DSPy represent an important advance in the field of prompt engineering optimisation. DSPy, developed by Khattab et al. (2023) and accessible via its GitHub repository[5], provides a framework for algorithmically optimising LLM prompts and weights, with the potential to automate and refine prompt-crafting processes. This represents a potentially transformative advance in the ongoing evolution of prompt engineering.

In conclusion, prompt engineering represents a fundamental technique that has been systematically applied and refined in this paper with the aim of enhancing the utility of LLMs in supporting process coherence in BPM. The incorporation of prompt engineering into the construction of aProCheCk was guided by the integration of established prompting techniques and adherence to best practices.