Agentic RAG for Case-Based Adaptation

Tolga Tel[1,*,†], Mirjam Minor[1,†]

[1]Goethe University, Robert-Mayer-Straße 10, 60325 Frankfurt, Germany

Abstract

Recently, multiple CBR-RAG approaches that retrieve cases from a case base in order to improve prompting have been investigated and achieve successful results. Few work has been done in the field of case-based adaptation in generative AI. Adaptation might contribute to mitigate flaws in generation results and, thus, to improve the output quality of generative AI.

In this paper, we present an agentic AI process including a novel, targeted case-based adaptation method for generated texts by a re-interpretation of knowledge-based adaptation rules. The solution approach uses co-working agents of different types within an agentic RAG system. The *evaluator agent* judges a generated solution in a two-step process. In the annotate step, the agent enriches the generation results using an LLM with structured outputs. In the validate step, the evaluator agent identifies mistakes in different error types, such as hallucinations or missing content. The second agent type is called *adaptation engineer*. It transforms the list of mistakes received from the evaluator agent into actionable feedback. Following the Case-based Reasoning (CBR) paradigm, the adaptation engineer selects matching adaptation rules from the knowledge base, creates adaptation instructions, and inserts the instructions into targeted prompts for each mistake. The *generator agent* uses an LLM to execute the prompts received from the adaptation engineer. The resulting, modified solution is sent again to the evaluator agent. With the communication between the agents, we created a more transparent and innovative solution for the description of graphs and diagrams.

We evaluate our adaptation approach for the sample task of describing business process models to end users. The experimental evaluation is conducted partly with the help of experts as well as automated using regular expressions. The experimental data comprises 28 business process models in business process model and notation (BPMN) as well as 15 adaptation rules specified by experts.

Keywords

Adaptation, Generative and Agentic AI, Agentic RAG, Validation

1. Introduction

Establishing trust in Large Language Models (LLMs) is critically dependent on reducing generative hallucination and guaranteeing output consistency. Current strategies, including sophisticated prompting techniques and Retrieval-Augmented Generation (RAG) [1] for contextual grounding, help to reduce these problems but do not provide a complete solution. There are some problems where the generated solution can be verified against the input query. To tackle these, we introduce a novel Agentic-RAG approach that incorporates a direct feedback loop to adapt the solution. Agentic-RAG is a novel RAG paradigm that "employs autonomous agents to orchestrate retrieval, filter relevant information, and refine responses, excelling in scenarios requiring precision and adaptability" [2].

We developed co-working agents for different tasks that can interact with each other following the Agentic-RAG structure. We orchestrate the agents in a pattern called the Evaluator-Optimizer workflow pattern [2]. We extend this pattern by a third agent type named an adaptation engineer. Further, we enrich the other agent types with knowledge-based methods inherited from case-based reasoning (CBR) [3]. For our test use-case of describing business process models, the advanced Agentic-RAG makes a key difference to past work like [4]. The paper investigated a *refine* approach using a RAG system. The RAG system initially generated a descriptive text for a process model using an in-context

ICCBR Workshop on Memory-Based Reasoning for Responsible GenAI at ICCBR2026, August 15, 2026, Bremen, Germany

*Corresponding author.

†These authors contributed equally.

tel@em.uni-frankfurt.de (T. Tel); minor@cs.uni-frankfurt.de (M. Minor)

http://wi.cs.uni-frankfurt.de (T. Tel); http://wi.cs.uni-frankfurt.de (M. Minor)

0009-0005-5012-8250 (T. Tel); 0000-0002-6592-631X (M. Minor)

example, then iteratively improved its solution by receiving additional in-context examples. A drawback of that method was the lack of gradual improvement, as the system opted to fully re-generate the text rather than incrementally refine it.

In extension to [5], the contribution of this paper is an adaptation approach. It develops a transparent targeted validation and adaptation method based on structural information. The previous work generates explanatory texts step-by-step using a CBR-RAG system and merges the outputs in the end.

Our experimental evaluation measures to what extent our advanced Agentic-RAG approach is able to outperform the CBR-RAG approach without adaptation. Additionally, the evaluator and adaptation agents are evaluated individually to support the design of our feedback loop. The experimental results are very promising, which makes us very confident that the advances Agentic-RAG approach will also be applicable to further generative tasks, such as explaining UML diagrams. Additionally we believe, that this approach can be added to previous techniques to improve their output even further.

The remainder of the paper is organized as follows. Related work is discussed in Section 2. The concept of adaptation is presented in Section 3. Implementation and Evaluation of our experiments are discussed in Section 4. Last but not least, a short conclusion is made in Section 5.

2. Related work

The Gao survey [1] about RAG systems describes the evolution of Retrieval-Augmented Generation paradigms, while showing a progress from simple to sophisticated approaches. Naïve RAG systems use a basic three-step process: indexing, retrieval and generation. This traditional approach often struggles with precision and recall, leading to inaccuracies. To fix these issues, Advanced RAG introduces pre- and post-retrieval processes to enhance retrieval quality. A step further, Modular RAG offers enhanced adaptability by allowing different strategies and components to be interchanged, leading to a greater versatility and improved performance.

A new paradigm in RAG, Agentic-RAG, is being described by Singh et. al [2] as a system that introduces autonomous agents to the RAG framework. The agents can perform various tasks like reflection, planning, tool use, and multi-agent collaboration to manage retrieval strategies and iteratively refine their understanding of a given context. This work provides a detailed taxonomy of Agentic-RAG architectures and highlights their key applications. The core characteristics include Autonomous decision-making, iterative refinement and workflow optimization. Despite its potential, Agentic-RAG faces several challenges that need to be addressed, such as coordination complexity, computational overhead and scalability limitations.

In the field of case-based reasoning, a couple of RAG-based approaches regarding case-based retrieval and adaptation have been published recently [4, 5, 6, 7].

Retrieval augmented generation can enhance LLM output by adding prior knowledge as context. This is useful for tasks that are knowledge intensive. In [7] this system is used for legal question-answering. Case-Based Reasoning can structure retrieval in RAG with retrieval, indexing, and similarity methods to give LLM queries relevant cases as context. This makes prompts richer and improves answer quality, especially in legal tasks, by ensuring similarity between questions and evidence. This approach does not apply adaptation in its methods.

The paper [6] explores using cases instead of simple text snippets in RAG, and prompting LLMs to use case-based reasoning. They tested four scenarios: (1) LLM adapting a given case, (2) LLM selecting and adapting a case, (3) LLM selecting two similar cases and combining them, and (4) LLM selecting the most similar and most different cases and combining them. Their results show that providing LLMs with cases benefits the accuracy. While including adaptation, this approach includes it only implicitly. The LLM decides about the adaptation it of a given case by itself. In our work, we utilize explicit adaptation instructions to specifically instruct the LLM to certain tasks.

Minor and Kaucher [4] introduce a process-oriented case-based reasoning approach, supported by the capabilities of retrieval augmented generation, was employed to generate coherent, informative textual descriptions for Business Process Models. Our approach aims to overcome shortcomings in the

proposed "Refine" strategy. Here instead of a general refine, which re-generates the entire text, we aim to instruct targeted changes to keep correct parts.

A recent extension of [4] considers structural knowledge when combining CBR with RAG [5]. Here, the structure of the process model was utilized for a guided generation approach to increase accuracy and consistency. In contrast, this work employs explicit adaptation knowledge in a feedback loop.

In a community paper [3] the authors describe open challenges and research directions regarding Generative Artificial Intelligence in the field of CBR. Our approach can be classified as guided generation while explicitly working on the open challenge "Domain and Adaptation Knowledge", where not re-generating the entire content but rather fixing only problematic parts is important.

3. Adaptation Concept

Current solution approaches for advanced RAG [1] include refinement processes for generated outputs. After a reflection, they allow to re-prompt the generation task along with intermediate generation results with the aim of improving the accuracy of the output. However, it has been observed that naïve re-prompting leads to re-generation instead of actually refining the previous output.

Case-based reasoning (CBR) has a long tradition in adaptation methods for case solutions [8]. In CBR, a case denotes a problem-solution pair that has been successfully used in the past. To solve a current problem, a retrieved case is modified to achieve a better fit with the current problem to be solved by reusing the case. A typical example of an adaptation rule in CBR is to adjust the price of an accommodation if an additional person shall be included in a hotel booking.

We interpret traditional CBR methods for retrieval and reuse in the light of Agentic-RAG as follows. A case comprises a pair of a user query and an LLM-generated output as a solution for this query. The cases are recorded in the data pool of a RAG system called case base. If a new user query arises, a retrieved case from the case base is used as an in-context sample to augment the prompt for the generate step. In addition, the generated output (solution) is adapted iteratively to improve its accuracy. An iteration includes a validation of the generated output to identify and localize adaptation needs, the selection of an appropriate adaptation rule, and the actual execution of the adaptation. The latter requires the creation of actionable feedback in the form of a prompt. The prompt contains an explicit adaptation instruction that refers to a specific part of the output generated during the previous iteration.

3.1. Agent design

The adaptation concept is elaborated in an agent design with co-working agents of three types depicted in Fig. 1. Here we differ between agents that use LLM-based applications, which have an LLM symbol like the generator agent, and agents, that use knowledge-based applications like the adaptation engineer agent. These are depicted using a brain-symbol. The evaluator agent utilizes both, LLM-based and knowledge-based applications. This feedback loop is repeated until a certain accuracy threshold is surpassed or a maximum iteration of the loop is reached.

The generator agent type uses LLM-based applications and creates a textual output in response to a user query. For the initial generation, it retrieves an in-context sample from the case base following a naïve RAG approach. The evaluator agent type reflects whether the generated output can be accepted, i.e. whether the output covers the entire user query and whether it does not suffer from hallucination. The first step of automated reflection is to annotate the generated text with a structured output. Structured output approaches [9] have been investigated to improve the consistency and overall quality of LLM outputs. Usually LLMs produce creative and diverse responses. If the goal is to use LLM outputs for traditional algorithms with strict input conditions, structured outputs can be used to set the output type of an LLM.

Next, the resulting structural elements are compared for validation with the structural meta-data derived from the query. Structural elements that occur in the query but are not covered by the generated output are considered missing. Structural elements in the generated output that do not have an appropriate element in the query are called hallucinations. Structural elements in the generated output

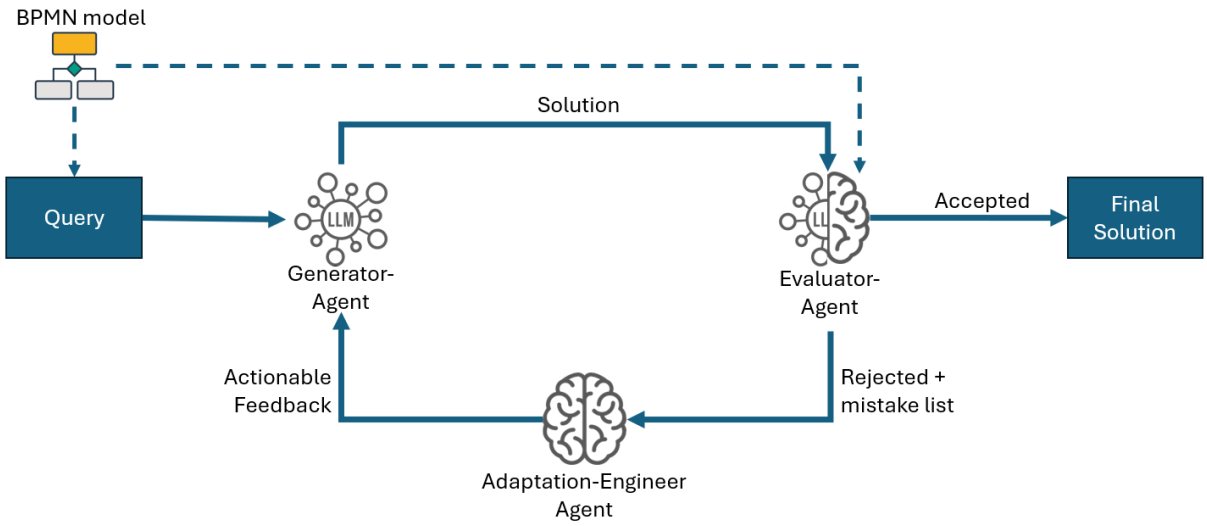


Figure 1: Agent types and interaction model.

that are slightly similar to an element that occurs in the query are called inaccuracies. All types of discrepancies, missing, hallucinated and inaccurate elements, are reported in a list of mistakes to the adaptation engineer. The third type of agent is named the adaptation engineer. It prepares the prompt instructions for the adaptation step. For each mistake on the list received from the evaluator, the adaptation engineer selects an appropriate adaptation rule from the adaptation knowledge base. Mistakes with the same adaptation rule are grouped together and filled in to a template for an adaptation instruction together with the instruction part of the adaptation rule. We consider the adaptation instructions as actionable feedback. They are sent step-by-step to the generator agent to be executed as parts of refine prompts. In contrast to refinement in conventional RAG systems, the adaptation instructions aim to modify only a targeted part of the previously generated output. The three types of agent are organized within an agentic-workflow comprising of generate-evaluate-adapt cycles.

3.2. Adaptation knowledge

Adaptation knowledge becomes necessary when new problems arise that have no exact matches in the case base. Adaptation knowledge can be represented in a variety of forms. One common method is through adaptation rules, which describe this knowledge using an IF-THEN structure. These rules are particularly well-suited for substitution tasks. A rule-based transformation specification can consist of one or more rules and may also include a termination criterion [10]. Formally, an adaptation rule can be written as follows:

$$IF(missing(El_1), missing(El_2), \ldots, missing(El_n))$$
$$THEN : insert(El_1, El_2, \ldots, El_n)$$

Here El_i are elements that are expected in the answer. For each category of elements this rule states, that all missing elements of that category should be inserted into the answer. Analogous superfluous elements can be eliminated by an adaptation rule with a delete instruction.

4. Implementation and evaluation

To validate our concept, we conducted an experiment on explaining business process models by adapting RAG-generated descriptive texts. The business process models are given in business model and notation (BPMN) stored as BPMN files. The adapted texts are compared to an initial text generated by a Naïve RAG as a baseline.

4.1. BPMN

The consortium OMG has established the standard BPMN [11] for modeling languages. BPMN offers a standardized graphical notation for visualizing business processes. This flowchart-like notation empowers stakeholders to design, manage, and realize business processes efficiently. Its independence from specific implementation environments ensures flexibility and adaptability.

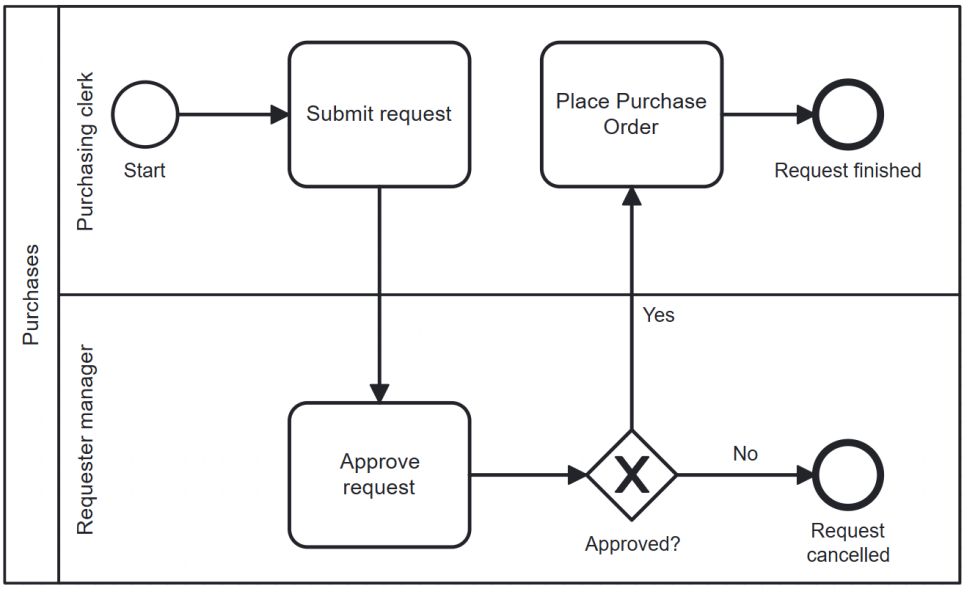


Figure 2: BPMN Example of purchase request approval.

An example for a small BPMN model can be found in Fig. 2. It describes the process of a purchase request approval, including three *tasks* ("Submit Request", "Approve Request", and "Place Purchase Order"). A *gateway* named "Approved?" controls the flow at a decision point, which allows branching depending on a decision. The circles represent *events*, where in this case the process starts or ends. The edges between these elements are called *sequence flows*. They define the order in which tasks and other elements are performed in a process. Last in this example we have lanes and pools, all elements are assigned to either the "Purchasing Clerk" or the "Requester Manager". The lanes are for different roles inside the same department or company depicted by a pool, which is called "Purchases" in the example.

By creating a common language, BPMN facilitates seamless communication between business and technical teams, streamlining the process of translating business requirements into executable software. This graphical notation of process models is saved in an XML-file that is machine-readable.

4.2. Implementation

The work is implemented in Python using the open-source AI orchestration framework Haystack [12]. The framework facilitates the development of LLM-based agents, multi-module applications and scalable RAG systems. Further, we use the Python library Pydantic for our structured output generation. It allows to transform the output of an LLM into a format that complies with the syntax requirements of a structured annotation template. The gpt-oss-20b model was selected as an LLM since it balances instruction-following capability, context length, and inference efficiency. Reproducible behavior is mostly achieved by using a very low temperature and a fixed random seed, which improves reproducibility across multiple experiments.

4.2.1. Generator Agent

The generator agent has two purposes. First, the initial generation of a solution text. Here we use a simple one-shot RAG system, that generates a descriptive text for a given process model. This RAG

system can be classified as a Naïve RAG.

Second, the adaptation of a generated text with given adaptation instructions. Here, the important change to other approaches, such as refine, is the targeted removal of errors without regenerating the entire descriptive text. It uses the prompt template in Fig. 3 completed by the adaptation engineer. The variable *mistake* is a placeholder for the particular adaptation needs (see below). Additionally, the corresponding process model and its textual description are provided in the prompt using the variables *bpmn_model* and *text*.

```
You are given a BPMN model and a corresponding descriptive text.
Given a BPMN model: {{ bpmn_model }}
With textual process description: {{ text }}
Correct only the following mistake in the description without
any formatting: {{ mistake }}
```

Figure 3: Prompt Template Adaptation.

An example descriptive text for the BPMN depicted in Fig. 2 is shown in Fig. 4. The description starts with a general overview of the BPMN model. Then each element is described in a short text.

```
The BPMN model describes a process titled Purchases. It involves two roles which are
the Purchasing clerk and the Requester manager. The workflow starts with a Start event
in the lane of the Purchasing clerk. The first action is for the clerk to Submit
request. Then the process moves to the Requester manager who must Approve request. The
outcome is determined by an exclusive gateway labeled Approved?. One path from the
gateway leads to a Request cancelled end event in the manager's lane. The other path
leads back to the Purchasing clerk to Place Purchase Order. If the order is placed the
process ends at the Request finished event.
```

Figure 4: Generated textual description for Fig.2.

4.2.2. Evaluator Agent

The evaluator agent receives the solution text from the generator agent and checks it for mistakes by comparing its contents with the existing contents of the process model.

```
{'properties': {'tasks': {'items': {'type': 'string'}, 'title': 'Tasks', 'type': 'array'},
  'events': {'items': {'type': 'string'}, 'title': 'Events', 'type': 'array'},
  'gateways': {'items': {'type': 'string'}, 'title': 'Gateways', 'type': 'array'},
  'sequence_flows': {'items': {'maxItems': 2, 'minItems': 2,
    'prefixItems': [{'type': 'string'}, {'type': 'string'}], 'type': 'array'},
   'title': 'Sequence Flows',
   'type': 'array'},
  'message_flows': {'items': {'maxItems': 2, 'minItems': 2,
    'prefixItems': [{'type': 'string'}, {'type': 'string'}], 'type': 'array'},
   'title': 'Message Flows',
   'type': 'array'},
  ...
}
```

Figure 5: JSON Schema excerpt for structured outputs.

In the *extract step*, the LLM is instructed to extract information from a descriptive text into a structured output format specified in JSON. Fig. 5 depicts the JSON schema for the particular BPMN elements,

including tasks, events, gateways, etc. as an input. The output is a JSON object in the given schema as seen in Fig.6.

```
{"tasks":["Submit request","Approve request","Place Purchase Order"],
"events":["Start event","Request cancelled end event","Request finished event"],
"gateways":["Approved?"],
"sequence_flows":[["Start event","Submit request"],
    ["Submit request","Approve request"],
    ["Approve request","Approved?"],
    ["Approved?","Request cancelled end event"],
    ["Approved?","Place Purchase Order"],
    ["Place Purchase Order","Request finished event"]],
"message_flows":[]}
```

Figure 6: Example structured output for a text describing Fig. 2.

In the *validate step*, we transform the XML file for the BPMN process into lists in the same JSON format as depicted in Fig. 5 algorithmically. An example is given in Fig. 7.

```
{'task_names': ['Submit request', 'Approve request', 'Place Purchase Order'],
 'event_names': ['Start', 'Request finished', 'Request cancelled'],
 'gateway_names': ['Approved?'],
 'flow_names': [['Start', 'Submit request'],
  ['Submit request', 'Approve request'],
  ['Approve request', 'Approved?'],
  ['Approved?', 'Request cancelled'],
  ['Approved?', 'Place Purchase Order'],
  ['Place Purchase Order', 'Request finished']],
 'msg_names': []}
```

Figure 7: Extracted information from XML-file represented in Fig. 2.

After comparing information from the solution text with the ground truth from the XML file, we classify each element into one of four categories: correct, missing, hallucinated or inaccurate. Elements are classified as missing, if they exist in the original XML file but not in the solution text. Correct elements are those found in both sources. Hallucinated elements exist in the solution text but not in the XML file. Inaccurate elements in the solution text are semantically similar to ones in the XML but not similar enough to be classified as correct. In our example, the only element that was not classified as correct. The event named 'Start' was extracted as 'Start Event' which was little below the acceptance threshold of our semantic similarity approach.

4.2.3. Adaptation Engineer

The adaptation engineer is implemented as a rule-based system that adds adaptation instructions to the prompt. A sample adaptation instruction for a missing tasks is depicted in Fig. 8. It implements the adaptation rule specified in Section 3.2 for tasks. The task name (x) is filled by means of the mistake list received from the evaluator agent.

The corresponding prompt template, where the instructions are added, is seen in Fig. 3. The variable *mistake* is filled with an adaptation instruction like the example in Fig. 8. This instruction can include removing hallucinated elements or adding missing elements.

The adaptation engineer works in a fixed order: first, the LLM is instructed to modify the text for the category 'inaccurate', then remove hallucinations. After giving instructions for modifying inaccurate

```
Task {x} is missing in the description of the process model.
Please add the task at the correct position.
```

Figure 8: Adaptation instruction for missing tasks.

elements and removing hallucinated elements, we instruct the LLM to add missing elements. We removed hallucinated elements before adding missing elements because this step reduces the text size, which increases the likelihood of success for future instructions by eliminating problematic content early on. The order of element types within the same category of mistake is as follows: (1) tasks, (2) gateways, (3) events, (4) sequence flows, (5) message flows. The corresponding prompt template, where the instructions are added, is seen in Fig. 3. In our running example, the only mistake found is the name of the event 'Start' which was named 'Start event' instead. This problem will be fixed by prompting a modify instruction, as seen in 9. The similarity of 'Start' and 'Start Event' is neither low enough to count as non-matching nor high enough to count as matches.

```
Event 'Start event' is similar to event 'Start'. Please change it in the description of
the process model to match event 'Start'.
```

Figure 9: Adaptation instruction for the example from Fig. 2.

Each mistake category and element type is handled in an own iteration of the generate-evaluate-adapt cycle. Further, the evaluator agent implements an additional break condition for the current iteration: If the text is worse than the last evaluated text, it is scrapped, and the mistake that was prompted in the last attempt is removed from the current iteration. If the text is at least as good as the previous text, an updated list of mistakes is then given to the adaptation engineer.

4.3. Experimental Evaluation

To evaluate our adaptation approach we use our adaptation pipeline for 28 different process models. Here a given initially generated text is first evaluated and then adapted if necessary. We prompt the correction of the mistake lists in two iterations, where in the second iteration all remaining mistakes, that are left after the first iteration, are reprompted. Since our adaptation approach consists of an evaluator agent and an adaptation agent, we will evaluate both separately. The adaptation agent will be evaluated by a human expert both end-to-end and after the first iteration. The evaluator agent will be evaluated by the accuracy of its evaluations on the generated texts by a human expert.

The metric used for comparison is the F1-score, which consists of precision and recall metrics. These metrics rely on counts of True Positives (TP), False Positives (FP) and True Negatives (TN). Precision measures the quality or correctness of a positive prediction. Here it answers the question: "Of all contents described by the model in the descriptive text, how many were actually correct?".

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

Recall measures the completeness of the model's positive predictions. Here it is the proportion of all actual elements of the element type in the process model that the model correctly identified. If a particular element type does not exist within a process model, the recall value is automatically set to 1, as the text has successfully named all (zero) existing elements of that type.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

The F1-score [13] is a combination of these two scores. It penalizes both, low precision and low recall, similarly and is the harmonic mean of *precision* and *recall*.

$$F1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

Business process models come in different sizes. Additionally, many types are only used rarely. To deal with these problems, we use two different metrics related to the F1-score to evaluate an average over all 28 BPMN models that were analyzed in this work.

Macro F1-score The macro F1-score [14] is calculated as an arithmetic mean of the F1 scores for each individual process model. This method treats all process models equally, regardless of size or if a specific element is included in this process model.

$$F1_{macro} = \frac{1}{N} \sum_{i=1}^{N} F1_i \tag{4}$$

Micro F1-score The micro F1-score [14] is different since it evaluates the overall effectiveness of the approach as if all descriptions were joined together for a single large process model. By first calculating micro-precision

$$\text{micro-precision} = \frac{\sum_{i=1}^{N} \text{TP}_i}{\sum_{i=1}^{N} \text{TP}_i + \sum_{i=1}^{N} \text{FP}_i} \tag{5}$$

and micro-recall

$$\text{micro-recall} = \frac{\sum_{i=1}^{N} \text{TP}_i}{\sum_{i=1}^{N} \text{TP}_i + \sum_{i=1}^{N} \text{FN}_i} \tag{6}$$

over all process models, the micro F1-score increases the impact of larger process models since they include more elements and are therefore more relevant in the total sum of TP, FP and FN.

$$\text{micro-}F1 = \frac{2 * \text{micro-precision} * \text{micro-recall}}{\text{micro-precision} + \text{micro-recall}}. \tag{7}$$

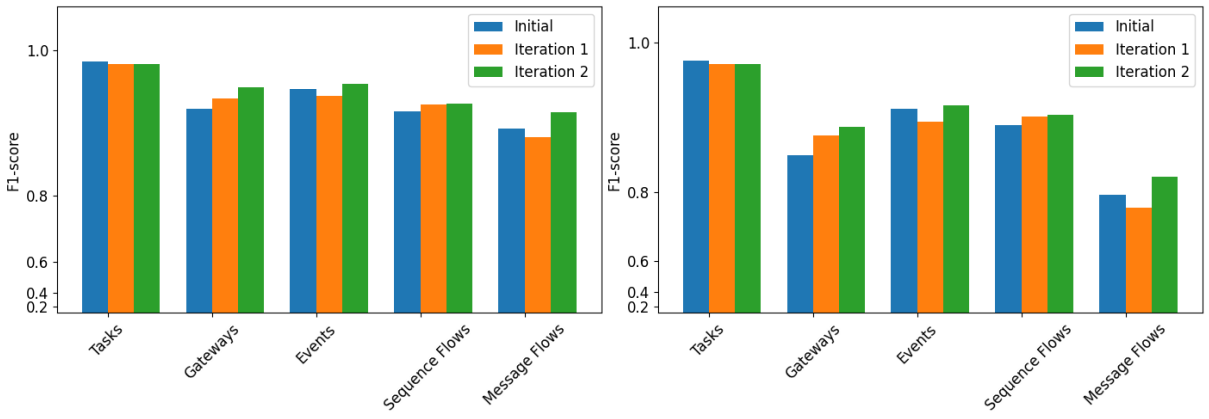
4.4. Experimental Results

We used the adapt pipeline for 28 different BPMN models. Here 19 BPMN models worked without errors after the first try, 5 more ran without errors in the second try. Out of these 24 models, where the pipeline ran without errors, 3 models had an initial text, where the validator agent correctly classified the text as completely correct and one model, where after the first iteration all mistakes were corrected.

4.4.1. Adaptation Performance

The experimental results for the adaptation approach show that the targeted adaptation instructions improve given descriptive texts of business process models for most element types.

We compare our adaptation approach against a Naïve RAG baseline. In this baseline, an LLM is given a process model as a BPMN file. The RAG system then retrieves an in-context example using classic similarity metrics. This example consists of another process model (also in BPMN format) and its corresponding descriptive text. This in-context example is also given to the LLM to generate a descriptive text. This strategy will be referred to as "Initial". Our "Adapt" strategy described in Sections 3 and 4 uses the texts generated by the "Initial" strategy as its foundation and is divided into two iterations.



(a) Macro F1-scores (log-scale).

(b) Micro F1-scores (log-scale).

Figure 10: Resulting F1 scores in log-scale.

Except for tasks, in both, micro- and macro-averaged F1-scores the adapt strategy outperforms the "Initial" strategy. This is visualized in Figures 10a and 10b. The slight decrease of the score in category "tasks" is due to a single task that has not been detected by the evaluator agent properly. The largest improvement in the F1-score was observed for the element type *gateways*. Here the Macro-F1 score changed from 0.936 to 0.961 and the Micro-F1 score changed from 0.864 to 0.906 after the second iteration. This is mostly due to an increase in recall. Nonetheless did the precision value increase to a perfect 1 after all hallucinations were removed in the first iteration.

Both sequence flows and message flows improve regarding their F1-score. Similar to the results for gateways, the recall values have a higher improvement than the precision. This is mostly due to the fact that there were almost no hallucinations in the initial texts. For sequence flows the Macro F1-score increases from 0.933 to 0.942 while the Micro F1-scores increase from 0.899 to 0.917 after the second iteration. For message flows the Macro F1-scores increase from 0.91 to 0.932 while the Micro F1-scores increase from 0.794 to 0.828 after the second iteration.

Analysis reveals that the minor reduction in F1-score performance of tasks was caused due to an error in the validator agent.

As depicted in Fig. 11 for both, gateways and events, we additionally separated each element type into unnamed and named elements to inspect their respective recall values individually. For both types, the named variants are described with a higher accuracy than the unnamed counterparts.

4.4.2. Performance of the Evaluator Agent

Since the detection of errors is vital to this adaptation approach, when evaluating this adaptation approach, a crucial factor is the performance of the evaluator agent. Here we use the precision and recall values of both, the evaluator and the human expert, to analyze differences in the evaluation.

An important observation is the abundance of false positives from the evaluator. Only in rare instances and only specifically in sequence flows, we observed false positives, where the human expert detected a sequence flow as missing while the evaluator extracted this flow from the text. Other than these rare occurrences, we can observe, that the evaluator is harsher than its human counterpart and often needs a more literal explanation of a BPMN element to extract it from the text.

The differences between human expert and the evaluator agent occur mostly due to unnamed elements as seen in Fig. 12b. Elements with names can be referenced more easily and therefore are easier to detect and add to the text if missing. The average difference between the evaluator and the human expert is decreasing across all observed element types in each iteration as seen in Fig. 12a which supports the assumption, that the texts become clearer in each iteration.

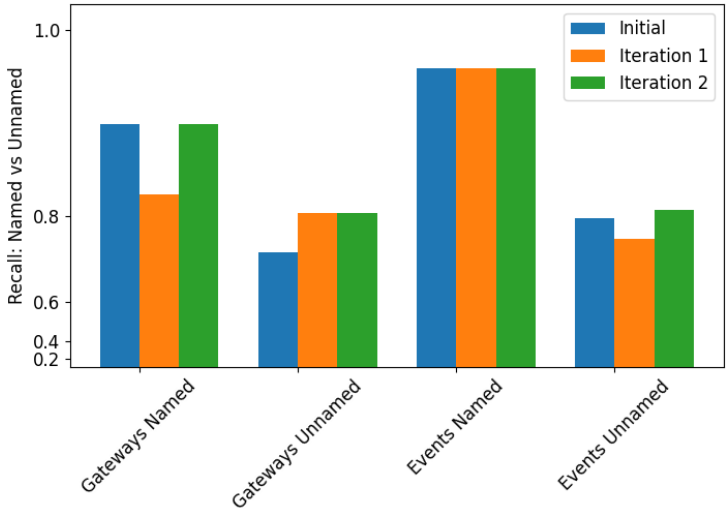
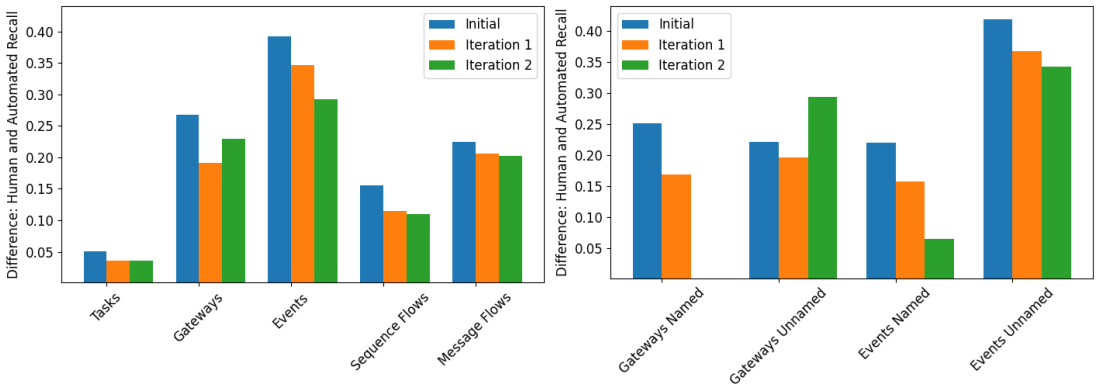


Figure 11: Micro Recall scores for named and unnamed gateways and events in log-scale.



(a) All different inspected element types. (b) named vs. unnamed gateways and events.

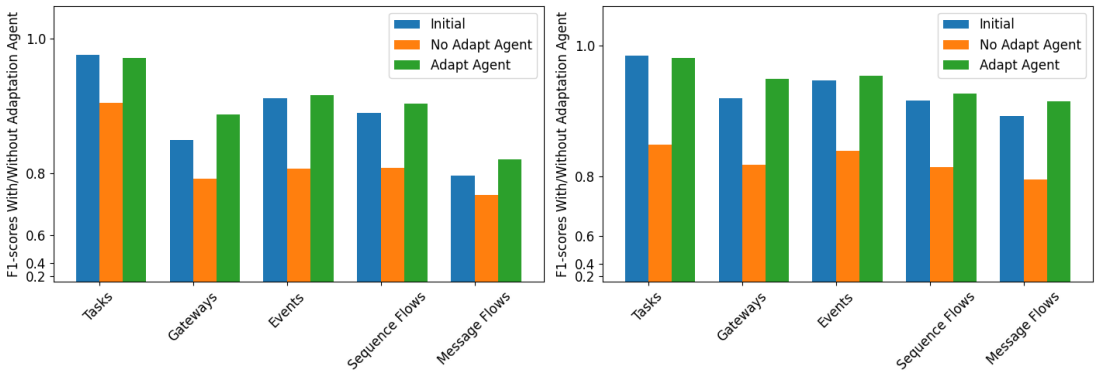
Figure 12: Weighted difference of evaluator agent and human expert.

4.4.3. Performance of the Adaptation-Engineer Agent

The adaptation engineer is a crucial part of our pipeline. To evaluate its influence, we ran all experiments without its actionable feedback. Here the evaluator agent passes the entire mistake list to the generator agent. The results in Fig. 13 show that the absence of the adaptation engineer decreases the F1-scores across all element types. For most textual descriptions the generated results were unchanged, because the evaluator agent identified the new texts as worse than the previously generated texts and withdrew them. In some cases, the adapted texts were entirely replaced by a list of all elements. There the evaluator agent did not identify the texts as worse because it is simply identifying occurrences. In the absence of the adaptation engineer no textual description was improved.

5. Conclusion and Future Work

This paper generally analyzed the applicability of an adaptation agent in the setting of process modeling. Our results confirm that a second iteration consistently optimizes text quality across nearly all element types, which validates the efficacy of our adaptation approach. Most notable were improvements regarding gateways. In many cases, the texts were also clarified in a way, that the automatic evaluation could detect the elements easier from text. The evaluator agent generally worked out great with its semantic comparison method. We see a clear difference between named and unnamed BPMN elements,



(a) Micro F1-scores.

(b) Macro F1-scores.

Figure 13: Scores with/without Adaptation Agent.

where named objects are easier to identify in texts for the evaluator agent and easier to include for the adaptation and generator agent. Unnamed elements also have a negative influence on the accuracy of sequence and message flows. Most of the time when a task, gateway or event is missing, all sequence flows and message flows, that start or end in it are considered as missing too. Our approach can be further elaborated by working with a hierarchical task structure, where for example the list of mistakes is first prompted including multiple mistakes at once and later broken down to increase granularity if needed. The hierarchical approach is also helpful regarding another problem. Currently, each mistake is prompted one by one, which can lead to higher runtime. If multiple mistakes are prompted at once, this could decrease runtime and the amount of tokens that need to be generated in the iterations. While comparing element names is already a promising approach, the next logical step would be reconstructing the entire BPMN graph from text to then use graph similarity functions for comparison. This approach also helps visualizing the extracted information and increase transparency of the evaluator agent. We demonstrate that case-based adaptation serves as a robust mechanism for improving generative AI in terms of de-hallucination and output consistency in applications where structural knowledge is available.

Acknowledgments

We would like to thank our student researcher Kim Linh Do for her helpful contribution to this paper. Her dedication to the Adaptation concept and meticulous attention to detail significantly contributed to the quality of the findings.

Declaration on Generative AI

The authors declare that no generative artificial intelligence (AI) tools were employed in the creation, writing, or editing of this manuscript.

References

[1] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, H. Wang, Retrieval-augmented generation for large language models: A survey, 2024. URL: https://arxiv.org/abs/2312.10997. arXiv:2312.10997.

[2] A. Singh, A. Ehtesham, S. Kumar, T. T. Khoei, Agentic retrieval-augmented generation: A survey on agentic RAG, 2025. URL: https://arxiv.org/abs/2501.09136. arXiv:2501.09136.

[3] K. Bach, R. Bergmann, F. Brand, M. Caro-Martínez, V. Eisenstadt, M. W. Floyd, L. Jayawardena, D. Leake, M. Lenz, L. Malburg, D. H. Ménager, M. Minor, B. Schack, I. Watson, K. Wilkerson, N. Wiratunga, Case-Based Reasoning Meets Large Language Models: A Research Manifesto For Open Challenges and Research Directions, 2025. URL: https://hal.science/hal-05006761, working paper or preprint.

[4] M. Minor, E. Kaucher, Retrieval augmented generation with LLMs for explaining business process models, in: International Conference on Case-Based Reasoning, Springer, Springer-Verlag, Merida, Mexico, 2024, pp. 175–190.

[5] T. Tel, M. Minor, Utilizing the Structure of Process Models for Guided Generation of Explanatory Texts, in: I. Bichindaritz, B. López (Eds.), Case-Based Reasoning Research and Development - 33rd International Conference, ICCBR 2025, Biarritz, France, June 30 - July 3, 2025, Proceedings, volume 15662 of *Lecture Notes in Computer Science*, Springer, Biarritz, France, 2025, pp. 157–171. URL: https://doi.org/10.1007/978-3-031-96559-3_11. doi:10.1007/978-3-031-96559-3_11.

[6] K. Wilkerson, D. Leake, On implementing case-based reasoning with large language models, in: International Conference on Case-Based Reasoning, Springer, Springer-Verlag, Merida, Mexico, 2024, pp. 404–417.

[7] N. Wiratunga, R. Abeyratne, L. Jayawardena, K. Martin, S. Massie, I. Nkisi-Orji, R. Weerasinghe, A. Liret, B. Fleisch, CBR-RAG: case-based reasoning for retrieval augmented generation in llms for legal question answering, in: International Conference on Case-Based Reasoning, Springer, Springer-Verlag, Merida, Mexico, 2024, pp. 445–460.

[8] R. Bergmann, M. Minor, K. Bach, K.-D. Althoff, H. Munoz-Avila, Fallbasiertes Schließen, in: Handbuch der Künstlichen Intelligenz, 6. auflage ed., De Gruyter, Berlin, 2020.

[9] M. X. Liu, F. Liu, A. J. Fiannaca, T. Koo, L. Dixon, M. Terry, C. J. Cai, "We Need Structured Output": Towards User-centered Constraints on Large Language Model Output, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, ACM, Honolulu HI USA, 2024, pp. 1–9. doi:10.1145/3613905.3650756.

[10] M. Richter, R. Weber, Case-based reasoning: a textbook, Springer, Berlin, 2013.

[11] O. M. Group, Business process model and notation, 2014. URL: https://www.omg.org/spec/BPMN/2.0.2#document-metadata.

[12] M. Pietsch, T. Möller, B. Kostic, J. Risch, M. Pippi, M. Jobanputra, S. Zanzottera, S. Cerza, V. Blagojevic, T. Stadelmann, T. Soni, S. Lee, Haystack: the end-to-end NLP framework for pragmatic builders, 2019. URL: https://github.com/deepset-ai/haystack.

[13] C. J. V. Rijsbergen, Information Retrieval, 2nd ed., Butterworth-Heinemann, Oxford, 1979.

[14] C. D. Manning, P. Raghavan, H. Schütze, Introduction to information retrieval, Cambridge university press, Cambridge, 2008.