Large Language Models as Knowledge Engineers

Florian Brand[1,2,*], Lukas Malburg[1,2] and Ralph Bergmann[1,2]

[1]Artificial Intelligence and Intelligent Information Systems, University of Trier, 54296 Trier, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, 54296 Trier, Germany

Abstract

Many Artificial Intelligence (AI) systems require human-engineered knowledge at their core to reason about new problems based on this knowledge, with Case-Based Reasoning (CBR) being no exception. However, the acquisition of this knowledge is a time-consuming and laborious task for the domain experts who provide the needed knowledge. We propose an approach to help in the creation of this knowledge by leveraging Large Language Models (LLMs) in conjunction with existing knowledge to create the vocabulary and case base for a complex real-world domain. We find that LLMs are capable of generating knowledge, with results improving by using natural language and instructions. Furthermore, permissively licensed models like CodeLlama and Mixtral perform similar or better than closed state-of-the-art models like GPT-3.5 Turbo and GPT-4 Turbo.

Keywords

Case-Based Reasoning, Knowledge Engineering, Knowledge Acquisition Bottleneck, Large Language Models, Prompting

1. Introduction

Various Artificial Intelligence (AI) systems, ranging from knowledge-based systems such as expert systems or AI planning, rely on knowledge, which is created, designed, and acquired by humans [1]. Case-Based Reasoning (CBR) [2] is no exception to this, with (human-engineered) knowledge being of importance throughout all phases of the CBR cycle. However, as this knowledge is reliant on domain experts, this knowledge is difficult to come by, a problem also known as knowledge engineering and acquisition bottleneck [3]. Existing work mostly focuses on the acquisition of adaptation knowledge [4, 5, 6, 7, 8], leaving the engineering and acquisition of the remaining knowledge to the domain expert. However, recent advancements in Natural Language Processing, particularly in LLMs, have led to various tools to aid domain experts in the creation of knowledge for different AI systems with promising results, particularly in the creation of planning domain descriptions for AI planning systems [9, 10, 11]. Nevertheless, these works focus on the usage of closed models and are currently only concerned about the knowledge creation process in AI planning and not CBR. This work answers the research question whether LLMs could be used to help in the acquisition of CBR-specific knowledge and how LLMs should be applied in practice to achieve this. We investigate the impact of different prompting techniques and the performance of various models, including closed models, as well as permissively licensed models. The paper is structured as follows: First, an overview about knowledge in CBR and LLMs is provided, followed by a discussion about related work (see Section 2). We then present our approach in Section 3 and show the implementation of this approach for a domain in Section 4. Section 5 shows the results of the experiments, with Section 5.4 concluding the paper and giving an outlook for future work.

2. Foundations and Related Work

In this section, we give an overview of the concept of knowledge in CBR based on the knowledge containers introduced by Richter [12] (see Section 2.1). Subsequently, we introduce Large Language

ICCBR CBR-LLM'24: Workshop on Case-Based Reasoning and Large Language Model Synergies at ICCBR2024, July 1, 2024, Mérida, Mexico
*Corresponding author.
brand@uni-trier.de (F. Brand); malburgl@uni-trier.de (L. Malburg); bergmann@uni-trier.de (R. Bergmann)
0000-0002-0877-7063 (F. Brand); 0000-0002-6866-0799 (L. Malburg); 0000-0002-5515-7158 (R. Bergmann)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3708/paper_01.pdf

Models in Section 2.2 and prompting in Section 2.3. Finally, the section closes with an overview of related work (see Section 2.4).

2.1. Knowledge in Case-Based Reasoning

The most prominent way to represent knowledge in a CBR system is to view knowledge in the form of knowledge containers, as proposed by Richter [12]:

Vocabulary Container: The vocabulary container stores knowledge about the description of the elements which describe the objects and the elements of the cases itself, for example the word "position" describing the locality of an object [12]. Therefore, the used vocabulary encodes a lot of knowledge and is the basic for any knowledge-based system. Furthermore, different descriptions can (and should) be used to describe objects depending on the given task.

Similarity Container: The similarity container consists of the knowledge needed to determine the similarity between two cases to approximate the utility to reuse the solution suitable for the new problem. There are different possibilities to calculate the similarity between cases, which range from simple symbolic similarities like the equality of two objects, to the usage of weighted similarity measures for complex objects. The similarity is used for the retrieval step in the CBR cycle and thus requires knowledge about the problem at hand as well as possible solutions [12].

Case Base Container: As the name implies, the case base container contains the experiences, i.e., the cases. The case base grows over time and can be created from experiences or be completely synthetic. This container is therefore the cornerstone of any CBR system and the main source of knowledge [12].

Adaptation Container: To adapt existing cases to new problems, the knowledge stored inside the adaptation container can be used. There are several algorithms to adapt cases to the given problem to adapt cases in a semi-automatic or fully automatic manner [13, 14, 7]. Consequently, the adaptation container stores the information needed to execute the given algorithm(s) in the required format.

2.2. Large Language Models

Language Modeling is one of the biggest areas of Natural Language Processing, with LLMs being the latest advancement in this area. While they can be fine-tuned to be suitable for any downstream task, these tasks can also be treated as a "text-to-text" problem, i.e., the model produces text as an output instead of the desired label for the downstream task [15]. This allows to apply the same model and training task for every downstream task imaginable and marks a paradigm shift. Instead of fine-tuning a model for a given downstream task, a sufficiently capable model just needs to be *prompted*, which in practice means to change the textual input to the model. The biggest advantage of prompting over fine-tuning is the ease of usage: Fine-tuning is a laborious task which requires the acquisition of a suitable training dataset and the training process itself, whereas a change in the prompt instantly results in a change of the output. Furthermore, prompting is computationally more effective than fine-tuning an existing model [16]. To make the prompting of LLMs easier for users, pre-trained models can be fine-tuned to follow instructions, which yields an *instruction-tuned LLM* [17, 18]. Besides the easier usage, instruction-tuned models are more controllable and significantly more preferred by users over the pre-trained base models [18, 16]. Furthermore, instruction-tuning improves the performance of the model over a range of benchmarks, including held-out ones [17].

2.3. Structuring of Prompts

As established in the previous section, prompting has become the preferred and superior usage pattern of LLMs. However, the exact wording of the prompt itself has severe implications to the performance of the model for the given task, which results in various techniques for prompting [19, 20, 21]. There exist several strategies for prompting, as seen in Figure 2.3. The simplest prompts are *zero-shot* prompts, in which the prompt has no additional details about the tasks and the model completes the given input:

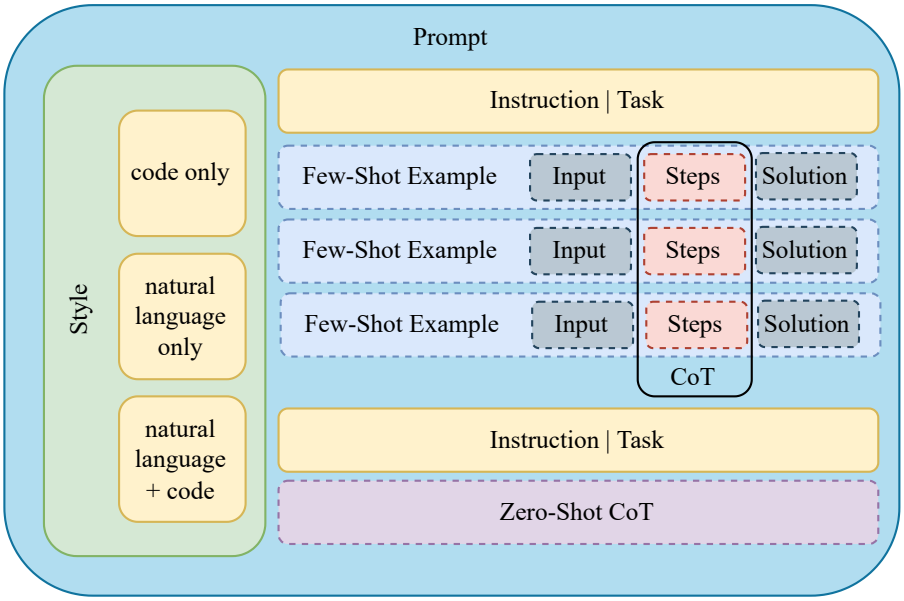

Figure 1: Different Prompting Strategies

Zero-Shot Prompt

Who wrote the book the origin of species?

Charles Darwin

In the above example, the first half of the box is the prompt, whereas the second half is the completion of the LLM.

LLMs are capable of *in-context learning*, which describes the conditioning of a model with natural language instructions and a few demonstrations, also referred to as *few-shot prompting* [22]:

In-Context Learning Prompt

Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>

---

fromage

The given prompt consists of a natural language instruction ("Translate English to French") and three examples, making it a three-shot prompt. In-context learning, a meta-learning technique, is similar to human intelligence, as argued by Lake et al. [23]. Humans can learn a new concept from only a single or a handful of examples, whereas many machine learning approaches need a lot of data points.

Building upon in-context learning, Chain-of-Thought (prompting) prompting results in improvements over a range of benchmarks [24]. In this technique, the model is provided not only the answer to a question, but as the steps to derive the answer as well:

Chain-of-Thought Prompt

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9.

Kojima et al. [25] introduce *zero-shot CoT*, a prompting technique which results in similar outputs to CoT prompting, without the need of the steps being provided. Their technique appends "Let's think step by step." to a prompt:

Zero-Shot Chain-of-Thought Prompt

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: Let's think step by step.

There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls.

While both CoT techniques improve the performance of LLMs over various benchmarks, the explanations provided by the models may be wrong, with the final answer of the output not following the reasoning of the explanation, indicating that the text and style of CoT is more important than the factual correctness of the output [26].

Besides the content of the prompt itself, the *style* of the prompt has implications on the performance, too. This can be divided into three categories: code-only, natural language only, or a mixture of both. Depending on the task at hand, the choice of styling is restricted. The code-only style obviously can only be applied in code-related tasks, such as the generation of SQL statements. These prompts can also be reworded into natural language, which intuitively should result in better results as LLMs are trained on a giant corpus of natural language. However, Sun et al. [27] found that prompting in a code-only style with a natural language instruction yields superior results over natural language prompts for generating SQL queries. A mixture of both is used by Zhang et al. [28] and Singh et al. [29], where natural language is used in a coding syntax, such as Python, without the code being executable or supplying necessary information:

Code-only Prompt

```
instructions = "Given a goal and two steps, predict the order to do the steps to achieve the goal"
goal = "Draw a Simple Teddy Bear"
step0 = "erase unnecessary lines"
step1 = "draw a shirt for the bear"
order_of_execution =
```

[step1, step0]

2.4. Related Work

LLMs have been successfully applied in the generation of knowledge in various domains. Guan et al. [9] and Oswald et al. [30] use LLMs to generate knowledge for AI planning methods, in particular PDDL domains. Both first translate actions from an existing domain from PDDL into natural language and task the LLM to re-create the given action for the domain. Liu et al. [10] use LLMs to generate PDDL problems by describing the problem in natural language and supplying an exemplary PDDL domain. Gestrin et al. [11] use LLMs to generate both a PDDL domain and the matching PDDL problem by supplying a task description in natural language. Furthermore, LLMs have also been successfully applied in the generation of process models in the form of BPMN[1], which is also an important topic in the field of Process-Oriented Case-Based Reasoning [31]. Kourani et al. [32] utilize LLMs with different prompting techniques to generate workflows from textual descriptions. Feedback from users can be used to iteratively refine the workflows, which can be transformed into Petri nets or BPMN models. Bernardi et al. [33] fine-tune LLMs to generate process models, with the prompts being further enhanced by retrieving relevant chunks of the BPMN representation format.

In CBR, the main focus lies upon the acquisition of adaptation knowledge, which is one of the hardest challenges in CBR [3]. To combat this, a wide range of methods to automatically adapt cases have been proposed for numerous domains [4, 5, 6, 7, 8]. These methods include algorithmic approaches, which utilize the case base to subsequently apply patterns onto new problems or generalize and specialize cases, to Machine Learning-based methods or an approach based on Reinforcement Learning. LLMs have not been applied in the creation of CBR knowledge, which will be addressed in this paper.

3. Engineering Knowledge for Case-Based Reasoning with Large Language Models

As laid out in the previous chapter, prompting LLMs is the superior approach for using LLMs. This is especially apparent in the process of creating new knowledge, where the data for the new domain is sparse or even non-existent. We present a general approach to create knowledge in CBR, which is domain-agnostic, and show the applicability of the approach in an applied domain. Figure 2 gives an overview of the approach. Knowledge, which can be in any format, is given as part of the prompt to the LLM, which then generates knowledge in the desired output format.

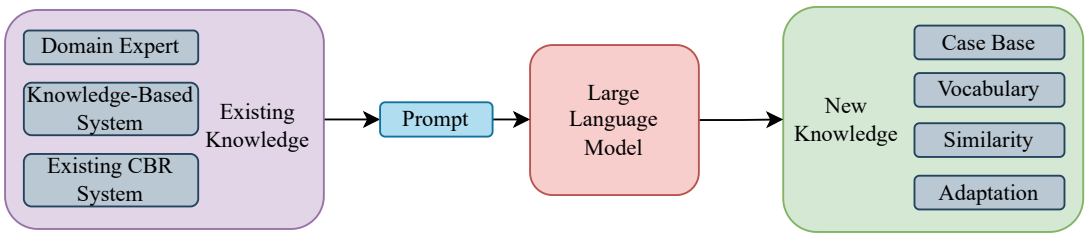

Figure 2: Generating knowledge with LLMs

The concept has four different parameters which are changeable: The new knowledge to be generated, the existing knowledge, the prompt, and the LLM.

New Knowledge: The knowledge to generate should represent the knowledge from any of the knowledge containers introduced in Section 2.1. We expect that every container is suitable to be generated by this method: The vocabulary container can be created and extended with a LLM, but also parts of the similarity container or the case base can be created. For the adaptation container, the LLM can be used to generate knowledge for any semi-automatic or fully automatic adaptation processes. The output from the LLM should be in a format which is directly usable by the CBR system, i.e., the

[1] https://www.omg.org/spec/BPMN/2.0.2/

LLM should generate knowledge in the form of code to be used by the CBR system, such as knowledge in the form of XML.

Existing Knowledge: The existing knowledge used for the prompt can be in various formats, stemming from different sources. For example, the prompt can be written entirely by domain experts, with the needed knowledge provided by them. Alternatively, a knowledge base can be queried, with the outputs being inserted into the prompt. It is also possible to use knowledge from an existing CBR system to either expand the CBR system or to use another CBR system from a similar domain as a reference point, similar to transfer learning. Furthermore, it is also possible to include multiple sources of knowledge: Knowledge from an existing CBR system can be used and supplemented with knowledge by a domain expert to further specify the desired goal.

Prompt: There exist several strategies to design the prompt itself, as shown in Section 2.3. However, not all aspects are useful in the context of prompting. Prompts for the generation of knowledge must involve few-shot examples which denote the input, i.e., existing knowledge, and the desired solution, i.e., the new knowledge from any knowledge container. Thereby, every prompt contains at least one example with existing knowledge and the desired target, e.g., a part of the vocabulary container.

In theory, a zero-shot prompt is possible, but the LLM would be unable to generate valid knowledge due to the lack of information about the scheme. Thus, zero-shot prompts are not a valid strategy for the generation of valid knowledge.

Zero-Shot Prompt Example

```
# Example 1
## Conditions:
- <Existing Knowledge>
Generate a valid CBR model.
```

Aside from the given examples, a prompt may include an instruction and use Zero-Shot CoT at the end of the prompt. For the style, natural language only is not useful, as the goal is to generate CBR knowledge directly, which is supplied as part of the few-shot example. The other styles encode the inserted knowledge either in the form of code or describe it using natural language.

Large Language Model: There are no special requirements for the LLM and thus, every LLM could possibly be used for the task. However, as the prompts themselves use some examples due to the few-shot setting, the LLM should have a bigger context size to fit the examples.

4. Exemplary Application Scenario for Generating Knowledge in a Cyber-Physical Domain

As a case study, the concept is implemented in the cyber-physical domain, which is concerned about the production of workpieces in a learning factory. The CBR system of the domain stores the processes to execute in the factory and adapts these in case a machine breaks. The subsequent sections introduce the domain and the deployed CBR system, followed by the implementation of the approach for this domain.

4.1. Introduction to the Cyber-Physical Domain

The smart factory, deployed at the University of Trier, is depicted in Figure 3. As an abstraction over a real-world factory, it can be used as a *Learning Factory* [35] that provides insights on a smaller scale, which can then be transferred to solve real-world problems. The factory at hand consists of two shop floors, which contain five identical workstations in a mirrored layout: A sorting machine with color detection, a high-bay warehouse, an oven connected to a milling through a workstation transport robot and a vacuum gripper robot. Additionally, there are machines unique to a singular shop floor: The first shop floor features a human workstation, which is capable of performing a range of possible

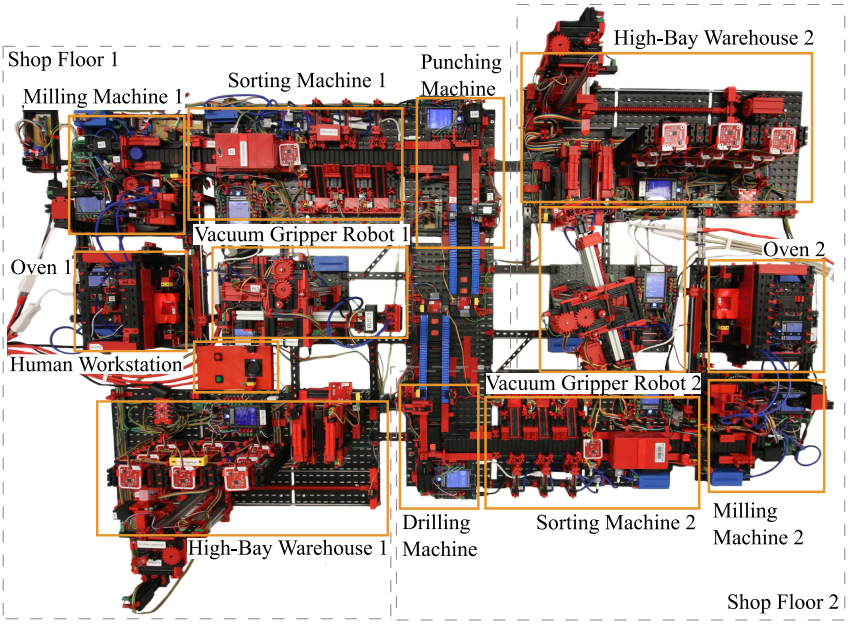

Figure 3: The Physical Smart Factory at the University of Trier (Source: [34])

tasks and a punching machine, while the second floor possess a drilling machine. The machines are connected through various conveyor belts, providing a route from the milling machine through the sorting machine to the drilling (or punching) machine, the vacuum gripper and the human workstation, which can reach almost all positions by simulating the transport done by a human. Furthermore, the machines are equipped with various sensors, including NFC readers/writers, light barriers, switches, and pressure sensors to provide insights about the current state of the machines [34]. To control the factory, a service-based approach is used, with each service representing multiple actions of one or multiple machines. As an example, a *transport* service combines several motor actuations and sensor observations into a single function. These services and the machines with their motors and sensors are modelled in an ontology [36]. Furthermore, the services are semantically enriched in the ontology with pre- and postconditions. For transporting a workpiece, the preconditions state that the machine needs to be ready to execute any service, a workpiece must lay at the starting position while the ending position is empty. These web services are also modeled in a CBR vocabulary in ProCAKE[2] [37], which supports

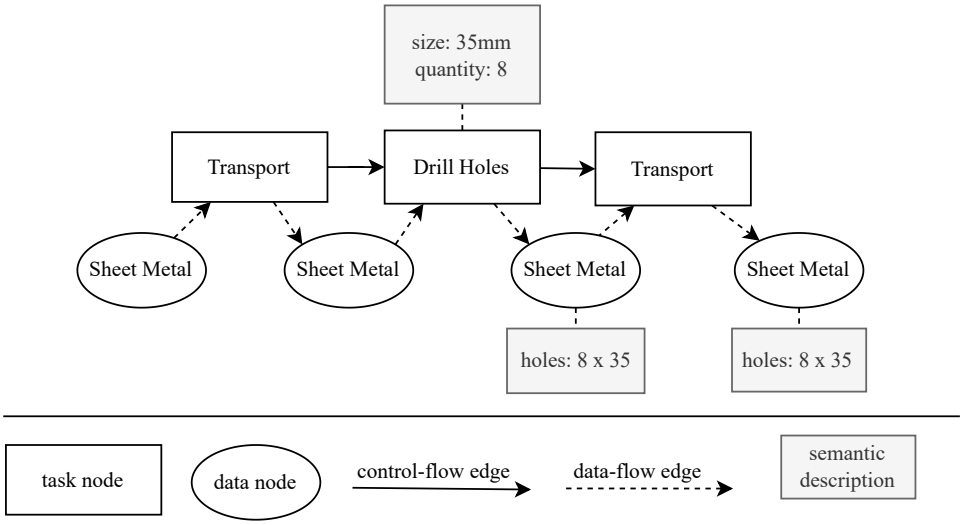

Figure 4: A Semantic Workflow Graph for Controlling the Smart Factory (Source: [5])

all knowledge containers [38]. The vocabulary container is represented by a domain model in ProCAKE. For the factory, the model, which is engineered entirely by domain experts, contains the representation of the workpiece properties to be manufactured and the basic properties of the physical factory, such as the positions of each machine. Furthermore, the semantic web services are described, albeit on a more general level. To control the factory, the services can be used as part of semantic graphs, which model not only the tasks to be executed in the factory, but also the state of the workpiece. Such a workflow is depicted in Figure 4. Task nodes, represented by white squares, denote the services to execute in the real-world factory. These task nodes represent, together with the semantic descriptors (gray squares), the semantic web services introduced earlier. Data nodes, shown as white circles, represent the state of the workpiece during this point in the workflow and are subsequently consumed and produced by each task node. The given semantic graph therefore shows the transport of a metal sheet, in which eight holes get drilled into, before it gets transported again.

4.2. Implementing the Knowledge-Generation Process

Algorithm 1 shows the overall algorithm that is applied to the selected application scenario, with the steps being described in more detail in subsequent sections.

Algorithm 1: Generate and validate knowledge from a LLM

Input : Examples E, LLM LLM and Prompt Template P

$used\_examples \leftarrow$ SELECT_RANDOM(E, n)
if *use_natural_language* **then**
  $used\_examples \leftarrow$ TRANSLATE(used_examples)
end
$prompt \leftarrow P \cup used\_examples$
$output \leftarrow LLM(prompt)$
$target \leftarrow$ EXTRACT_TARGET(output)
VERIFY_TARGET(target)

4.3. Selection of LLMs

For the implementation, different LLMs, which are considered state-of-the-art during the time of writing, are used. The following models are selected:
GPT-3.5 Turbo from OpenAI, specifically `gpt-3.5-turbo-1106`.
GPT-4 Turbo [39] from OpenAI, specifically `gpt-4-1106-preview`, which is considered state-of-the-art at the time of writing.
CodeLlama [40] from META, which is the best permissively licensed coding model. The evaluation uses two variants: The raw, pre-trained version, referred to as CodeLlama, and the instruction-tuned version, referred to as CodeLlama-Instruct. Both models are used in their 34B parameter variant.
Mixtral [41], which is considered the best open, general model at the time of writing. The instruction-tuned variant, `Mixtral-8x7B-Instruct-v0.1` is used.

4.4. Selecting and Translating Examples

At first, *n* examples are chosen from the knowledge base, i.e., semantic web services from the ontology or semantic graphs from the case base with *n* being the number of shots in the prompt, so a 5-shot prompt uses 5 examples, whereas a 3-shot prompt uses 3 examples. To retrieve the knowledge from the ontology, SPARQL [42] queries are used. SPARQL is a query language with a SQL-like syntax to retrieve information from an ontology or to even construct a new RDF graph. If a web service is retrieved, the matching representation of this web service in the CBR model is supplied to the prompt as well. As an

[2] https://procake.uni-trier.de

optional step, the knowledge can be translated into natural language to change the style of the prompt, similar to the approach by Guan et al. [9].

```
(?Name = vgr_transport_to_pm_1_sink_pos)
(?Type = <#preconditionsImplyOverAll>)
(?subjectName = VGR_1) (?predicateName = isReady)
(?objectName = true) (?paramValue = )

(?Type = <#postconditionsImplyAtEnd>)
(?subjectName = BusinessKey) (?predicateName = at)
(?objectName = pm_1_sink_pos) (?paramValue = pm_1_sink_pos)
```

Listing 1: SPARQL result of a query. Names shortened for brevity.

```
Over all, the VGR_1 should be ready.
At the end, be at the pm_1_sink_pos.
The name of the service is vgr_transport_to_pm_1_sink_pos
```

Listing 2: SPARQL result of a query

Listing 1 shows an excerpt of the results of a SPARQL query of the ontology, whereas Listing 2 shows the same information, but in natural language. The same web service as modeled in the vocabulary container of the CBR model is shown in Listing 3.

```
<StringClass name="end"
  superClass="ShopFloorPositions">
  <InstanceEnumerationPredicate>
    <Value v="pm_1_sink_pos"/>
  </InstanceEnumerationPredicate>
</StringClass>
<AggregateClass name="vgr_transport_to_pm_1_sink_pos">
  <Property name="url">/vgr/transport_to_pm_1_sink_pos</Property>
    <Property name="outputProperties">
      <Property name="position" value="pm_1_sink_pos"/>
    </Property>
  </Property>
</AggregateClass>
```

Listing 3: CBR representation of the query from the previous Listings

As mentioned in Section 2, the targets are not translated, as the LLM-generated output should be in the format of the knowledge, i.e., directly usable XML for the CBR model. However, the workflows stored in the case base are an exception to this rule: In ProCAKE, cases are stored in XML and result in many tokens for the prompts when using the chosen models from Section 4.3. Therefore, workflows from the case base are converted into a custom format based on YAML. This format encodes the task nodes of a cyber-physical workflow and the data flow of the workflow. The workflow from Figure 4 as represented in YAML is shown in Listing 4.

```
1. Transport:
    production_step: vgr_transport
    parameters:
        resource: vgr_2
        start: ov_2_pos
        end: dm_2_pos
2. Drill Holes:
    production_step: dm_drill
    parameters:
```

```
            resource: dm_2
            start: dm_2_pos
            end: dm_2_sink_pos
            quantity: 8
            size: 35
    3. Transport:
        production_step: vgr_transport
        parameters:
            resource: vgr_2
            start: dm_2_sink_pos
            end: hbw_2_pos
```

Listing 4: A NEST Workflow in YAML

The conversion of NEST graphs from XML to YAML has substantial effects: A single workflow with 10 tasks encoded in XML uses roughly 10,000 tokens with the (Code)Llama and Mixtral tokenizer and 8,000 tokens when using the tokenizer for GPT-3.5 and GPT-4. Converting the workflow into YAML results in roughly 1,300 tokens for Llama and Mixtral, whereas GPT-based models use 1,000 tokens. A similar effect can be observed when translating the ontology into natural language, which results in roughly 75% less tokens.

4.5. Prompt Templates

As shown in Section 3, the acquisition of knowledge can be prompted in different styles, by (not) using instructions and by (not) utilizing CoT. Therefore, there are three different prompting styles, with each style being represented by a template.

The basic template uses no instructions and directly inserts the examples.

Prompt with no Instructions for the CBR model

```
# Example 1
## Conditions:
- <Knowledge from Ontology>
## Resulting CBR model:
<CBR model in XML>

<Name of the service to generate>
## Conditions:
- <Knowledge from Ontology to generate>
## Resulting CBR model:
```

The template for the instruction prompt adds an instruction add the beginning of the prompt ("Your task is to generate a CBR model for the given conditions"), whereas the CoT prompt contains the instruction at the beginning and zero-shot CoT from Wei et al. [24] ("Let's think step by step").[3]

For the case base, workflow(s) are supplied in the prompt template.

Prompt with no Instructions for the case base

```
# Example 1
<Workflow in YAML>
# New Workflow:
```

[3]All prompts are also in the repository at https://gitlab.rlp.net/iot-lab-uni-trier/iccbr-2024-cbr-llm-workshop

4.6. Verifying LLM Outputs

To evaluate the output from the LLM, the relevant target, i.e., the YAML workflow or the CBR model, is extracted. This is done automatically by using regular expressions to find the relevant part(s). If the extraction fails, the target is extracted manually. Then, the target is syntactically and semantically validated using software-based verifiers. A software-based verifier gets the expected, gold label target and the LLM-generated output, i.e., the CBR model or the workflow, and then checks for syntactic or semantic equality. For the CBR model, the syntactic equality is checked by using the ProCAKE parser, whereas the semantic equality is checked by using XMLUnit[4]. The YAML workflows are first converted into their XML representation and then verified by ProCAKE parsers for syntactic and semantic equality.

5. Evaluation

The implementation from the previous section is evaluated using an experiment. Section 5.1 introduces the setup for the experiment, with Section 5.2 and Section 5.3 showing the results for the generation of the vocabulary container and the case base, respectively. Section 5.4 discusses the results.

5.1. Setup

The overall setup follows the implementation from the previous Section. To assess the impact of prompts and various parameters, different configurations for the prompts are used.

Prompt Template

As introduced in Chapter 6, there are three different prompt templates, which use either no instructions, use instructions and use instructions and CoT.

Number of Shots

Furthermore, the number of examples is varied for the prompts. The examples stay the same across all prompts, so every 5-shot prompt uses the same five examples every time.

Usage of Natural Language

For the CBR model, the conditions from the ontology are either inserted directly or translated into natural language. It is necessary to test all variations of the possible input variables to meaningfully assess the impact of each variable. For example, if testing the impact of using natural language, it is necessary to also test 1-, 3- and 5-shot prompts to assess the impact of natural language in various settings, mitigating a possible bias in the selection of a specific prompt technique. Finally, this setup is applied for testing every parameter described in the following.

For the evaluation, each combination is run 5 times due to the probabilistic nature of LLM outputs. We used 0.2 for the temperature and do not vary this parameter, as preliminary experiments showed a negligible impact when changing it.

5.2. Generation of a Vocabulary Container for CBR

Figure 5 shows the impact of using natural language as the input compared to using the SPARQL outputs directly. It can be seen that using natural language results in more syntactically and semantically correct outputs, which is true for all models, including the coding-focused CodeLlama models. This could be explained by the fact that CodeLlama is Llama 2, a general-purpose LLM, which is then further trained on code samples. Overall, the GPT models outperform the permissively licensed models, which becomes especially apparent when not using natural language, where Mixtral can only generate 53.33% semantically valid outputs.

Figure 6 shows the impact of the different prompting templates as introduced in Section 4.5. With the exception of GPT-4, using instructions in the prompt results in significant better results, even for the base, non-instruction-tuned CodeLlama. The CodeLlama models also show the biggest boost in performance when using instructions, with the results increasing by 70%. Interestingly, using CoT

[4]https://www.xmlunit.org/

results in a worse performance for every model, both syntactically and semantically. For all models, using no instructions at all yields better results than using CoT.

The number of examples, as depicted in Figure 7, has little impact on the overall performance, except for Mixtral, which is unable to generate valid CBR models with a single example. When using multiple examples, the performance greatly improves.

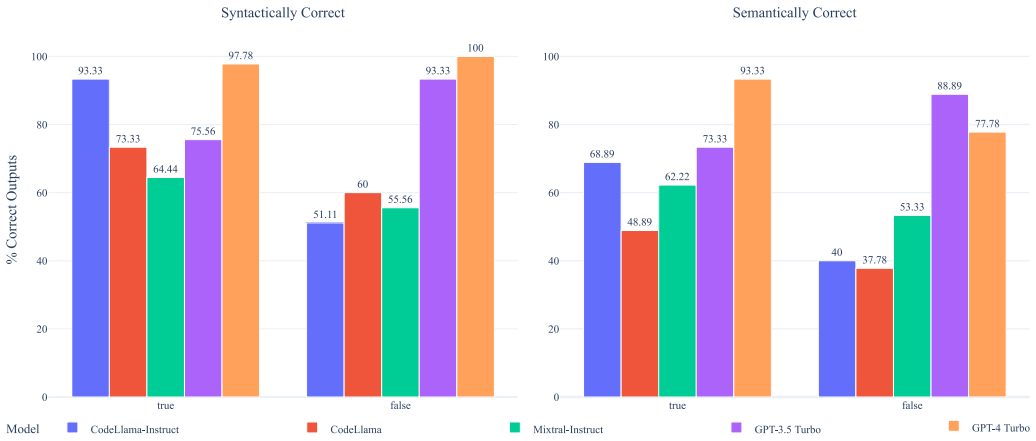

Figure 5: Generating the vocabulary container by using natural language

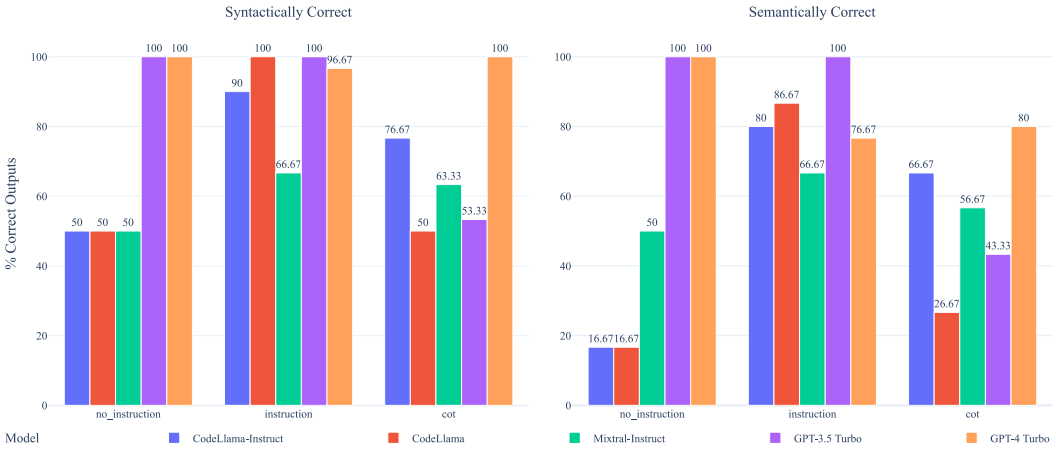

Figure 6: Generating the vocabulary container with different prompt styles

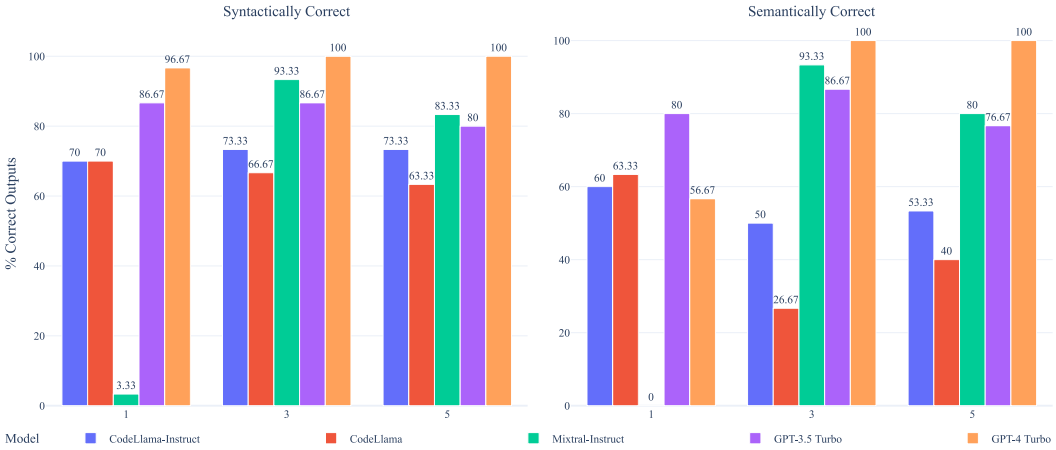

Figure 7: Generating the vocabulary container with a different number of examples

5.3. Generation of a Case Base for CBR

For the case base, using no instructions results in the best performance, as can be seen in Figure 8. The most surprising result is the performance of GPT-4 Turbo, which is unable to generate a single semantically valid example. Looking at the outputs from GPT-4 Turbo, the outputs fall into roughly two categories: The LLM output is the request for more clarifications or the output is *too creative*, hallucinating machines and services which do not exist, such as a non-existent "cooling station". A possible explanation for this result could lie in the instruction-tuning and RLHF phase of the training phase of GPT-4 Turbo, which could be focused on human conversations and creativity, resulting in those outputs for this task.

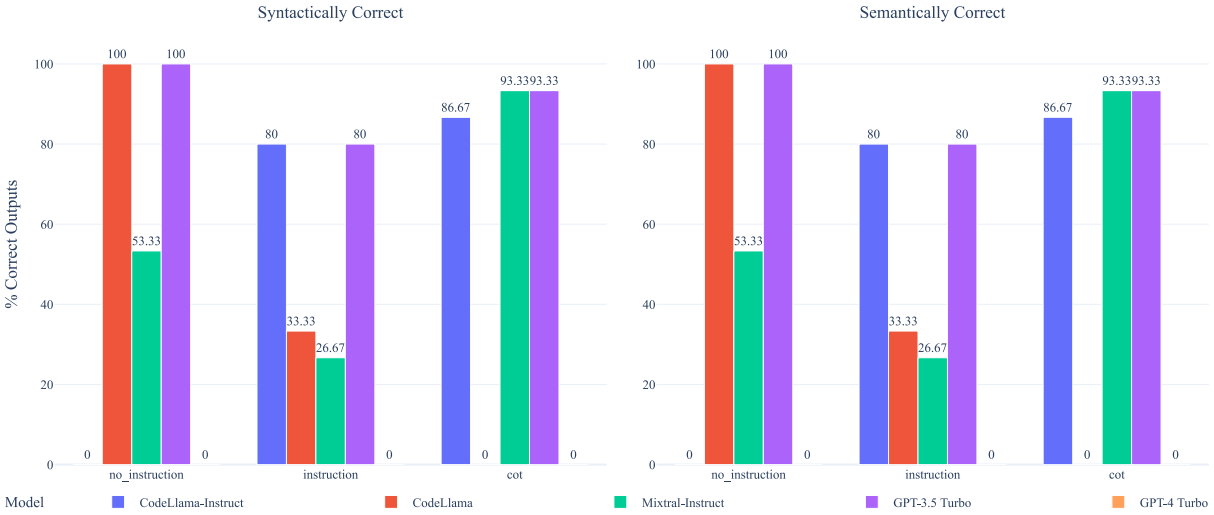

Figure 8: Generating cases with different prompt styles

The number of examples has impacts on the performance, as shown in Figure 9. More examples results in better performance for GPT-3.5 Turbo and Mixtral, whereas both CodeLlama variants perform worse when using three examples. Parallel to the other results, GPT-4 Turbo is unable to generate valid example cases.

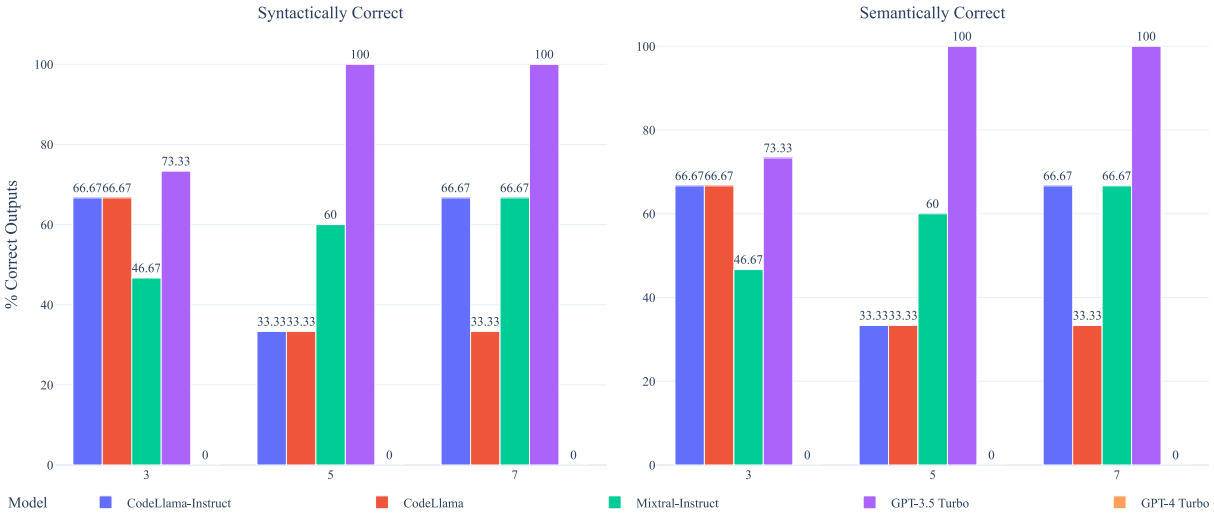

Figure 9: Generating cases with a different number of examples

5.4. Discussion

The results from the previous sections give numerous insights. Overall, it is mostly possible for LLMs to generate knowledge for the cyber-physical domain. As the approach is general and not specific for the domain, we expect the results to be transferable to other domains as well. This assumption on transferability of the general approach has already been experimentally demonstrated in other settings, such as AI planning [9, 43]. As a prerequisite for this, the domain needs at least some hand-engineered knowledge to be used as example(s) in the given prompts. The usage of natural language improves the results across multiple tasks, especially when the wording of the prompts is meaningful relative to the modeled domain and not obfuscated, as shown by similar works [9, 43, 11]. This also applies to CodeLlama, a coding-focused model, in which the base model being a general purpose LLM likely plays a role. Furthermore, adding instructions to prompts result in improved performance, whereas the usage of zero-shot CoT gives mixed results, highlighting the need of experimenting with different prompting techniques when creating knowledge. The number of examples in the prompt plays an important role, albeit a smaller one. However, the creation of the vocabulary container contained similar examples. When supplying the prompt with similar examples while the task is the creation of vocabulary which is dissimilar, the performance drops significantly with models being unable to successfully create the target[5]. Furthermore, permissively licensed models are close to the performance of closed models, even surpassing the performance when generating further examples for a case base. However, the evaluation is dependent on the software-based verifier, which can only check for equality due to their respective implementations, resulting in overly harsh results due to the binary feedback provided by the verifiers. When inspecting failure cases manually, numerous failures can be fixed by a human rather fast, as most errors are wrong variables and missing/superfluous XML tags.

6. Conclusion and Future Work

We present an approach using Large Language Models (LLMs) to generate knowledge for Case-Based Reasoning (CBR) systems. Similar to related work [10, 9], we propose the utilization of existing knowledge with examples to generate new, but similar knowledge ready-to-use in CBR. Our findings indicate that LLMs are capable of generating knowledge in this setting, but there exists a high variance in the results based on the prompting techniques used. Furthermore, permissively licensed models are capable of generating knowledge with a similar or even better performance than closed models.

For future work, different domains beside the used cyber-physical domain should be investigated. In this context, we examine further, more practice-oriented application scenarios in which LLMs can help to remedy the high efforts for knowledge acquisition and engineering. At a technical standpoint, different prompting strategies can be explored with a focus on the selection of the provided examples. It could also be researched whether it is possible to create knowledge without the need of providing (many) examples in the prompt itself. Moreover, there is also a trend emerging to eliminate the need to manually craft prompt with frameworks such as DSPy [44]. Furthermore, an interesting direction to research is to incorporate the feedback from the software-based verifiers into the knowledge-creation process. One possible approach to achieve this is to prompt the LLM with the initial prompt, the initial completion of the LLM, and the feedback from the verifier, thus making a multi-turn conversation, similar to the back prompting approach by Guan et al. [9]. Furthermore, our case study uses two of the four knowledge containers. While the approach proposed is general, future work could research the applicability to the similarity and adaptation containers. Finally, we want to integrate a LLM into the ProCAKE framework [37], helping CBR users to develop more easily customized CBR applications by adding a chat-based assistant to guide domain experts through the process of knowledge acquisition and modeling. Moreover, a LLM could augment existing CBR domains by using the existing vocabulary or case base to extend or generate new knowledge. In this context, a LLM can help users to generate suitable adaptation rules [45] in a domain, which is often a laborious and time-consuming task.

[5]All results are provided in the repository (https://gitlab.rlp.net/iot-lab-uni-trier/iccbr-2024-cbr-llm-workshop)

References

[1] Shu-Hsien Liao, Expert System Methodologies and Applications—a Decade Review from 1995 to 2004, Expert Systems with Applications 28 (2005) 93–103.

[2] A. Aamodt, E. Plaza, Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches 7 (1994) 39–59.

[3] K. Hanney, M. T. Keane, The Adaptation Knowledge Bottleneck: How to Ease it by Learning from Cases, in: 2nd ICCBR, Springer, 1997, pp. 359–370.

[4] R. Bergmann, G. Müller, Similarity-Based Retrieval and Automatic Adaptation of Semantic Workflows, in: Synergies Between Knowledge Engineering and Software Engineering, Advances in Intelligent Systems and Computing, Springer, 2018, pp. 31–54.

[5] F. Brand, K. Lott, L. Malburg, et al., Using Deep Reinforcement Learning for the Adaptation of Semantic Workflows, in: 31st ICCBR Workshops, volume 3438, CEUR-WS.org, 2023, pp. 55–70.

[6] L. Malburg, F. Brand, R. Bergmann, Adaptive Management of Cyber-Physical Workflows by Means of Case-Based Reasoning and Automated Planning, in: 26th EDOC Workshops, LNBIP, Springer, 2023, pp. 79–95.

[7] X. Ye, D. Leake, V. Jalali, et al., Learning Adaptations for Case-Based Classification: A Neural Network Approach, in: 29th ICCBR, volume 12877 of *LNCS*, Springer, 2021, pp. 279–293.

[8] C. Zeyen, L. Malburg, R. Bergmann, Adaptation of Scientific Workflows by Means of Process-Oriented Case-Based Reasoning, in: 27th ICCBR, volume 11680 of *LNCS*, Springer, 2019, pp. 388–403.

[9] L. Guan, K. Valmeekam, S. Sreedharan, et al., Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning, 2023. arXiv:2305.14909.

[10] B. Liu, Y. Jiang, X. Zhang, et al., LLM+P: Empowering Large Language Models with Optimal Planning Proficiency, 2023. arXiv:2304.11477.

[11] E. Gestrin, M. Kuhlmann, J. Seipp, NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions, 2024. arXiv:2405.04215.

[12] M. M. Richter, R. O. Weber, Case-Based Reasoning: A Textbook, Springer, 2013.

[13] R. Bergmann, Experience Management: Foundations, Development Methodology, and Internet-Based Applications, volume 2432 of *LNCS*, Springer, 2002.

[14] G. Müller, Workflow Modeling Assistance by Case-based Reasoning, Springer, 2018.

[15] C. Raffel, N. Shazeer, A. Roberts, et al., Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, Journal of Machine Learning Research 21 (2020) 1–67.

[16] S. Zhang, L. Dong, X. Li, et al., Instruction Tuning for Large Language Models: A Survey, 2023. arXiv:2308.10792.

[17] J. Wei, M. Bosma, V. Y. Zhao, et al., Finetuned Language Models Are Zero-Shot Learners, in: 10th ICLR, OpenReview.net, 2022.

[18] L. Ouyang, J. Wu, X. Jiang, et al., Training Language Models to Follow Instructions with Human Feedback, 2022. arXiv:2203.02155.

[19] P. Liu, W. Yuan, J. Fu, et al., Pre-Train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing, ACM Computing Surveys 55 (2023) 1–35.

[20] Y. Lu, M. Bartolo, A. Moore, et al., Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity, in: 60th ACL, ACL, 2022, pp. 8086–8098.

[21] T. Shin, Y. Razeghi, R. L. Logan IV, et al., AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts, in: EMNLP Conference, ACL, 2020, pp. 4222–4235.

[22] T. B. Brown, B. Mann, N. Ryder, et al., Language Models Are Few-Shot Learners, in: NeurIPS Conference, 2020.

[23] B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, Human-Level Concept Learning through Probabilistic Program Induction, Science 350 (2015) 1332–1338.

[24] J. Wei, X. Wang, D. Schuurmans, et al., Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, 2023. arXiv:2201.11903.

[25] T. Kojima, S. S. Gu, M. Reid, et al., Large Language Models Are Zero-Shot Reasoners, 2023.

arXiv:2205.11916.
[26] A. Madaan, A. Yazdanbakhsh, Text and Patterns: For Effective Chain of Thought, It Takes Two to Tango, 2022. arXiv:2209.07686.
[27] R. Sun, S. O. Arik, H. Nakhost, et al., SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL, 2023. arXiv:2306.00739.
[28] L. Zhang, L. Dugan, H. Xu, et al., Exploring the Curious Case of Code Prompts, 2023. arXiv:2304.13250.
[29] I. Singh, V. Blukis, A. Mousavian, et al., ProgPrompt: Generating Situated Robot Task Plans Using Large Language Models, 2022. arXiv:2209.11302.
[30] J. Oswald, K. Srinivas, H. Kokel, et al., Large Language Models as Planning Domain Generators (Student Abstract), in: AAAI Conference on Artificial Intelligence, 2024.
[31] M. Minor, S. Montani, J. A. Recio-García, Process-oriented case-based reasoning, Information Systems 40 (2014) 103–105.
[32] H. Kourani, A. Berti, D. Schuster, et al., ProMoAI: Process Modeling with Generative AI, 2024. arXiv:2403.04327.
[33] M. L. Bernardi, A. Casciani, M. Cimitile, et al., Conversing with Business Process-Aware Large Language Models: The BPLLM Framework, Preprint, In Review, 2024.
[34] L. Malburg, R. Seiger, R. Bergmann, B. Weber, Using Physical Factory Simulation Models for Business Process Management Research, volume 397 of *LNBIP*, Springer, 2020.
[35] E. Abele, G. Chryssolouris, W. Sihn, et al., Learning Factories for Future Oriented Research and Education in Manufacturing, CIRP Annals 66 (2017) 803–826.
[36] L. Malburg, P. Klein, R. Bergmann, Converting semantic web services into formal planning domain descriptions to enable manufacturing process planning and scheduling in industry 4.0, EAAI 126 (2023) 106727.
[37] R. Bergmann, L. Grumbach, L. Malburg, et al., ProCAKE: A Process-Oriented Case-Based Reasoning Framework, in: 27th ICCBR Workshops, volume 2567, CEUR-WS.org, 2019, pp. 156–161.
[38] A. Schultheis, C. Zeyen, R. Bergmann, An Overview and Comparison of Case-Based Reasoning Frameworks, in: 31st ICCBR, volume 14141, Springer, 2023, pp. 327–343.
[39] OpenAI, New Models and Developer Products Announced at DevDay, https://openai.com/blog/new-models-and-developer-products-announced-at-devday, 2023.
[40] B. Rozière, J. Gehring, F. Gloeckle, et al., Code Llama: Open Foundation Models for Code, arXiv (2023). arXiv:2308.12950.
[41] A. Q. Jiang, A. Sablayrolles, A. Roux, et al., Mixtral of Experts, 2024. arXiv:2401.04088.
[42] SPARQL Query Language for RDF, https://www.w3.org/TR/rdf-sparql-query/, 2008.
[43] K. Valmeekam, M. Marquez, S. Sreedharan, et al., On the Planning Abilities of Large Language Models – A Critical Investigation, 2023. arXiv:2305.15771.
[44] O. Khattab, A. Singhvi, P. Maheshwari, et al., DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines, 2023. arXiv:2310.03714.
[45] L. Malburg, M. Hotz, R. Bergmann, Improving Complex Adaptations in Process-Oriented Case-Based Reasoning by Applying Rule-Based Adaptation, in: 32nd ICCBR, Lecture Notes in Computer Science, Springer, 2024. Accepted for Publication.