Interpreting Workflow Architectures by LLMs

Michal Töpfer[a], Tomáš Bureš[b], František Plášil[c] and Petr Hnětynka[d]
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic

Keywords: Software Architecture, Large Language Model, LLM, Workflow, Workflow Architecture.

Abstract: In this paper, we focus on how reliably can a Large Lanuage Model (LLM) interpret a software architecture, namely a workflow architecture (WA). Even though our initial experiments show that an LLM can answer specific questions about a WA, it is unclear how correct its answers are. To this end, we propose a methodology to assess whether an LLM can correctly interpret a WA specification. Based on the conjecture that the LLM needs to correctly answer low-abstraction level questions to answer questions at a higher abstraction level properly, we define a set of test patterns, each of them providing a template for low-abstraction level questions, together with a metric for evaluating the correctness of LLM's answers. We posit that having this metric will allow us not only to establish which LLM model works the best with WAs, but also to determine what their concrete syntax and concepts are suitable to strengthen the correctness of LLM's interpretability of WA specifications. We demonstrate the methodology on the workflow specification language developed for a currently running Horizon Europe project.

1 INTRODUCTION

Large language models (LLMs) are being explored in many disciplines. This is also true for software architecture design, where LLMs are starting to be explored to guide, review, and generate software architectures.

In software architectures, as in other fields, a problem with LLMs is the reliability of their answers. In our experiments, we have seen that LLMs may provide very accurate answers that are almost completely hallucinated just from the names of components. This typically happens in cases when the architecture follows some well-established reference architecture rules.

It is not just the choice of a particular LLM that determines the correctness of answers, but it also matters how the LLM is queried and how the information given to it is represented. In the case of software architectures, the key information given to the LLM is software architecture specification—the obvious representative here being textual specification in an architecture description language (ADL).

We posit that if we could determine the correctness of an LLM's answers given the concrete syntax of the ADL, its semantics (including features like component nesting, different control/data-flow mechanisms, etc.), and the selection and form of mandatory metadata (e.g., component names, description of component behavior), we could design an LLM-friendly ADL, which would strike the trade-off between its expressiveness and its correct interpretability by the LLM.

As the first step in this endeavor, we focus on the problem of how to determine the correctness of an LLM's answers given a certain ADL. In this paper, we narrow this goal to a particular class of software architectures, namely workflow architectures (WAs). These are typically represented as an oriented graph of tasks and operators (e.g., branching, fork/join) bound by control and data flow links. Examples include languages such as BPMN[1]. We aim to target the above-mentioned problem by addressing the following research questions.

(RQ0): How to validate that an LLM can correctly interpret WAs and is not just hallucinating its answers?

(RQ1): How to measure the correctness of its interpretation?

To answer these questions, we outline a methodology for testing the quality (correctness) of LLM's interpretation of WAs.

The methodology is based on a set of parameter-

[a] https://orcid.org/0000-0002-3313-1766
[b] https://orcid.org/0000-0003-3622-9918
[c] https://orcid.org/0000-0003-1910-8989
[d] https://orcid.org/0000-0002-1008-6886

[1] https://www.bpmn.org/

Töpfer, M., Bureš, T., Plášil, F. and Hnětynka, P.
Interpreting Workflow Architectures by LLMs.
DOI: 10.5220/0013358000003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 608-617
ISBN: 978-989-758-742-9; ISSN: 2184-4895

ized patterns for creating low-abstraction-level questions from which particular test questions will be derived for a specific workflow ADL. The idea of using these patterns stems from the conjecture that successful answering of these questions is a necessary condition for reliably answering questions at a higher abstraction level (such as recommending a task from a repository to enhance the workflow's functionality in a desired way). Technically, we structure the patterns into categories reflecting key facets of the workflow specification (structure, behavior, basic functionality).

The structure of the paper is as follows. Sect. 2 shows a running example. In Sect. 3, the methodology is presented. Sect. 4 presents results and a related discussion. Sect. 5 summarizes the related work and Sect. 6 concludes the paper.

2 RUNNING EXAMPLE

To give a realistic grounding to our methodology, we demonstrate it on the workflow ADL (WADL further on) developed in the Horizon Europe project ExtremeXP[2], which focuses on modeling experiments via workflows.

Figure 1 shows a simple workflow BinaryClassification for training a classifier, specified as a graph (right) and WADL (left). Since we believe the basics of a WADL specification are easy to grasp, we only briefly comment on its key concepts.

Each workflow is defined by a block that starts with the workflow keyword followed by the workflow name. Within an individual workflow, the core entities are *task*, *data*, and *flow links*.

A task is defined by the task keyword (a round-corner rectangle in the graphical notation) and represents an action to be performed. A task can be either primitive (executable file, service, etc.—as TrainTestSplit on line 7) or is accomplished by a sub-workflow like ModelTraining (line 10).

The data (the data keyword, grey rectangles with a folded corner) serve as inputs and outputs of tasks and also of whole workflows.

By flow links, both *control flow* and *data flow* are defined (when a property relates to both of these concepts, we simply refer to a *flow*). A control flow (simple/solid arrows) defines the order in which the tasks are executed. It can contain branches via conditional links (an arrow plus question mark followed by a condition (as in line 21); an arrow starting with a diamond in graphical notation). There are other possibilities for more complex branching (such as parallel branching, for simplicity omitted in the example). The sequence of actions in a control flow is its *trace*. Finally, a data flow (double arrows in WADL and dashed arrows in graphical notation) defines where data are produced and consumed.

3 METHODOLOGY

To address both research questions, we designate a set of *test instances* for evaluating whether the LLM correctly interprets a WA specification in an ADL. Test instances are generated from *test patterns*, each testing a particular semantical aspect of the ADL (test instances' variability is supported by test patterns' parametrization).

The key elements of a test pattern are (i) a particular WA specified in an ADL, (ii) a question to LLM about WA, and (iii) a metric to evaluate the correctness of LLM's response (typically a reference answer).

3.1 Test Patterns

We devised the test patterns by systematically analyzing entities in the workflow meta-model developed within the ExtremeXP project. As mentioned in Sect. 1 we group the test patterns into the following three categories (listed by their increasing complexity): Structure (Sect. 3.1.1), Behavior (Sect. 3.1.2), Basic functionality (Sect. 3.1.3).

Even though their list is formally neither exhaustive nor complete, it currently reflects the main concerns we have encountered so far when experimenting with WADL and LLMs.

The following sections provide an overview and examples of the test patterns and Table 1 lists all the patterns. More details can be found in the replication package[3].

3.1.1 Structure

In this category, we focus on tasks, flow links, special link types (e.g., conditional control flow links), dependencies, and hierarchical structure in/of workflows.

For example, by applying a test pattern of this category, one can verify whether an LLM is able to correctly say if a specific task follows directly after another in a control flow specification (like TrainTestSplit and BinaryClassificationTraining – line 20 in Figure 1) or whether specific tasks are involved in a loop (BinaryClassificationTraining and BinaryClassificationEvaluation – line 20 and 21).

[2]https://extremexp.eu/

[3] https://github.com/smartarch/extremexp-llm/

```
1  // file BinaryClassification.wadl
2  workflow BinaryClassification {
3    description "Training of a binary classification ML model with the given
           hyperparameters.";
4    // tasks
5    task TrainTestSplit {
6      description "Splits data...";
7      implementation "file://split.py";
8      param test_size = 0.2;
9    }
10   task BinaryClassificationTraining {
11     subworkflow ModelTraining;
12   }
13   task BinaryClassificationEvaluation {...}
14   // data
15   data HyperParameters;
16   data InputData; data TrainingData; data TestData;
17   data MLModel; data MLModelMetrics;
18
19   // control flow
20   START –> TrainTestSplit –> BinaryClassificationTraining –>
           BinaryClassificationEvaluation;
21   BinaryClassificationEvaluation ?–> BinaryClassificationTraining
           { condition.../*retraining necessary*/; }
22   BinaryClassificationEvaluation ?–> END
           { condition... /*otherwise*/; }
23
24   // data flow
25   InputData ––> TrainTestSplit ––> TrainingData ––>
           BinaryClassificationTraining ––> MLModel ––>
           BinaryClassificationEvaluation ––> MLModelMetrics;
26   HyperParameters ––> HyperParameters;
27   TrainTestSplit ––> TestData ––> BinaryClassificationEvaluation;
28 }
29 ––––––––––––––––––––––––––––––––––––––––––––––––––––––
30 // file ModelTraining.wadl
31 workflow ModelTraining {
32   description "..."
33   // tasks
34   task FeatureExtraction {...}
35   task ModelInit {...}
36   task ModelFitting {...}
37   // data
38   data Hyperparameters;
39   data TrainingData; data TrainingFeatures;
40   data UntrainedMLModel; data MLModel;
41
42   // control flow
43   START –> FeatureExtraction –> ModelInit –> ModelFitting –> END;
44
45   // data flow
46   TrainingData ––> FeatureExtraction ––> TrainingFeatures ––> ModelFitting
           ––> MLModel;
47   HyperParameters ––> ModelInit;
48   HyperParameters ––> ModelFitting;
49   ModelInit ––> UntrainedMLModel ––> ModelFitting;
50 }
```

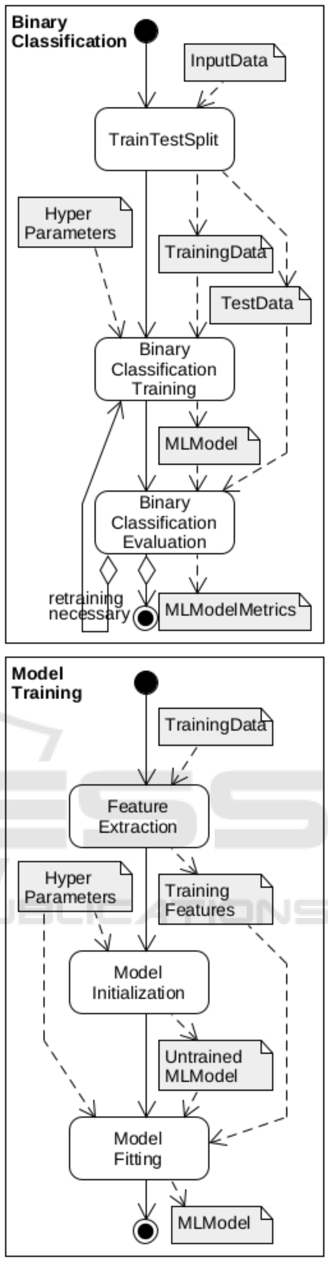

Figure 1: An excerpt of a workflow specification—WADL and graph.

3.1.2 Behavior

In this category, we focus on task execution order, conditional flow links, and traces.

For example, by applying a test pattern of this category, one can verify that the LLM can assess whether a loop in a workflow may end (as if the loop in Figure 1 ends when retraining necessary on line 21 is false) or whether a conditional branch could be taken.

3.1.3 Basic Functionality

In this category, we focus on task functionality, workflow functionality, and semantical order of tasks.

For example, using a test pattern of this category one can verify that the LLM is able to spot mistakes in the task order (e.g., whether the order of tasks BinaryClassificationTraining and BinaryClassificationEvaluation was mistakenly swapped in Figure 1).

Table 1: List of test patterns (grouped by category).

Category and Sub-category		**Pattern name**
Structure	Tasks	List of tasks
		List of tasks with a property (filter tasks)
	Flow links	Link existence
		Task after task
		Next tasks in flow
		Flow cycle
		Flow start detection
		Flow end detection
		Missing link *(flow is not continuous)*
	Special link types	Link existence with a property
		List of links with a property (filter links)
	Links operators	Operator existence (e.g., fork, join)
		Parallel tasks (block) existence
		List tasks in a parallel (fork-join) block
		Parallel tasks to a specific task
		List all parallel tasks
		List tasks in an operator block (other than simple fork-join)
	Depen-dencies	Dependency existence (in a flow)
		List of dependencies (in a flow)
		Data production (which task produces specific data)
	Hierar-chical structure	Task hierarchy
		List of composite tasks
		List of nested tasks
		Infinite recursion in references
Behavior	Task order	Are given tasks in correct order (corresponding to control flow)? (without conditional flow)
		Are all tasks in correct order? (without conditional flow)
		Determine task order (i.e., topological sort; without conditional flow)
	Condi-tional (control) flow	Is conditional flow mutually exclusive
		Are given tasks in correct order? (with conditional flow)
		Are all tasks in correct order? (with conditional flow)
		Next task in conditional flow
		Determine task order (i.e., topological sort; with conditional flow)
		Is loop infinite?
		Loop end condition
	Traces	Can trace of tasks occur with initial situation
		Does task run in every situation?
Basic functionality	Tasks function-ality	Describe task functionality (based on name, parameters, links, etc.)
		Inconsistent task name and description
		Inconsistent task name and other entities (e.g., data, linked tasks)
		Meaning (functionality) of tasks (e.g., task performs an operation that is not directly men-tioned in its name)
	Workflow function-ality	Describe workflow functionality
		Inconsistent workflow name and description
		Inconsistent descriptions of workflow and tasks
	Task order	Semantically incorrect order of tasks
		Meaning (functionality) of preceding tasks

3.2 Test Pattern Examples

In this section, we present three “representative” test patterns, each belonging to one of the three categories.

3.2.1 List of Tasks

Category: Structure

Rationale: In a workflow, can the LLM list the tasks with a given property?

Parameters: P, N, T, W
P: property of the tasks (e.g., task has a parameter),
N: number of tasks satisfying P,
T: total number of tasks,
W: workflow name

Architecture: Workflow W with N tasks satisfying P and $(T - N)$ tasks not satisfying P.

Question: List all tasks in workflow W that satisfy P.

Reference Answer: *A set of N tasks satisfying P*

Evaluation Metric: Jaccard index of the LLM’s answer set and the reference answer set[4]

Example of Instance:
Architecture: Figure 1,
Question: “List all tasks in workflow BinaryClassification that are realized by a sub-workflow.”,
Reference Answer: { BinaryClassificationTraining }

3.2.2 Mutually Exclusive Conditional Flow

Category: Behavior

Rationale: Can the LLM interpret conditional flow?

Parameters: F, W, T_0, T_1, T_2, C
F: flow link type (control flow or data flow),
W: workflow name,
T_0, T_1, T_2: tasks in the workflow,
C: condition for conditional link (in flow F)

Architecture: Workflow W with tasks T_0, T_1, T_2 (and possibly other), conditional link in flow F between T_0 and T_1 with condition C, conditional link in flow F between T_0 and T_2 with condition $\neg C$.

Question: Are conditional links in flow F from task T_0 mutually exclusive?

Reference Answer: yes

Evaluation Metric: correctness (1 if LLM’s answer is correct, 0 otherwise)

Example of Instance:
Architecture: Figure 1,
Question: “Are conditional control flow links from task BinaryClassificationEvaluation mutually exclusive?”
Reference Answer: yes (conditions “retraining necessary” (line 21) and “otherwise” (line 22) are mutually exclusive)

A similar pattern with the same question and reference answer “no” exists to also cover negative examples (links that are not mutually exclusive).

3.2.3 Inconsistent Task’s Name and Its Description

Category: Basic functionality

Rationale: Can the LLM detect inconsistency in a task name and task description?

Parameters: W, T, D_T
W: workflow name,
T: task name,
D_T: task description that is inconsistent with name T

Architecture: Workflow W with task T that has a description D_T.

Question: Identify tasks with inconsistency of their names and descriptions in W and present a list of them.

Reference Answer: The description of task T does not correspond to its name *(exact formulation might depend on the test instance)*.

Evaluation Metric: ROUGE-1 recall[5] or BERTScore[6]

Example of Instance:
Architecture: Task BinaryClassificationTraining (line 10) with description “Training of a *regression* ML model” (not shown in Figure 1)
Question: “Identify tasks with inconsistency of their names and descriptions in BinaryClassification and present a list of them.”
Reference Answer: “BinaryClassificationTraining: the task’s description Training of a regression ML model is inconsistent with its name indicating training of a binary classification model.”

4 RESULTS

To assess the viability of our methodology, we applied it to workflows from the ExtremeXP Horizon Europe

[4]The Jaccard index is defined as the size of the intersection divided by the size of the union of the sets. This penalizes both missing items in the LLM’s response and items that should not appear there.

[5]The ROUGE metric (Recall-Oriented Understudy for Gisting Evaluation) (Lin, 2004) determines the word overlap of the reference answer and the LLM output.

[6]BERTScore (Zhang et al., 2020) computes the cosine similarity of word embeddings (that capture the meaning of words)

project (a snippet is in Figure 1). For the initial evaluation, we instantiated only a subset of the test patterns, totaling 25 test instances (Table 2).

As part of the contribution, we developed a tool based on the LangChain Python framework[7] that allows to run the experiments automatically. The tool takes a list of instantiated patterns and constructs prompts from them, queries the LLMs (via the OpenAI API platform[8]), and evaluates the answers. Implementation, experiment setup, instantiated patterns, and raw results are available in the replication package[3].

4.1 Experiment Setup

For experiments, we used two LLMs: GPT-4o (version gpt-4o-2024-05-13) and GPT-4 Turbo (gpt-4-0125-preview). We evaluated how well these LLMs interpret WADL when employing three variants of its expressiveness: In the first one, each (sub)workflow is specified in a separate file. Since a workflow can be derived from (import) another workflow, it may be necessary to import more than one file to specify the workflow (*With imports WADL*). The second variant is simplified by inlining the importing effect (*Inlined WADL*). All sub-workflows are inlined in the third variant (*No sub-workflows WADL*).

For GPT-4o, we applied two modes of presenting the WADL specifications to the LLM: 1. In the *up-front* mode, all relevant WADL specifications are presented in the initial prompt, while 2. in *agent* mode, the LLM has to ask for the WADL specifications (sub-workflows) it needs. For GPT-4 Turbo, we only tested the agent mode. For more details, refer to Section 4.2.

The experiment was carried out by prompting the test instances to an LLM one at a time and then scoring the LLM's response by the pattern's metric.

4.2 Prompt Construction

During the evaluation, each test instance was treated as a separate LLM conversation, so they did not influence each other. The prompt encompassing a test instance of any test pattern consisted of four parts:

1. *A basic description* of the WADL expressiveness variant,
2. *Reference to the workflow* in agent mode, or *WADL specifications* in up-front mode (see below),
3. *Instructions* to the LLM (see below),
4. *Question*.

In terms of reference to the workflow in agent mode, we do not include the workflow specifications directly in the prompt. Rather, the LLM can request to read the workflow specification files it needs (via function calling[9]). We chose this approach because we wanted to see whether the LLM was able to figure out which further information was necessary (a workflow may contain references to other workflows).

In the up-front mode, we include the complete WADL specifications of all relevant workflows and sub-workflows in the initial prompt (instead of just providing a reference).

We instruct the LLM to think step by step (as recommended in the literature, e.g., by (Kojima et al., 2023), and known as the *chain of thought*) and write a short explanation before giving the final answer. In the agent mode, we also instruct to use function calls (tools) to obtain the required WADL specifications. In addition, we instruct to provide the answer in a specific form, e.g., the answer has to be "yes" or "no".

A prompt formulation is illustrated in Figure 2 and others are in the replication package[3].

4.3 Summary of Results

The results are summarized in Table 2 showing the average scores of the test instances for each pattern considered. Since LLM responses are stochastic, the prompting of each test instance was done five times. Thus, the table summarizes each score by *mean*. The "total" scores are averaging the scores of test instances in each pattern category.

Observing the scores and manually evaluating the actual LLM's responses, we have concluded that both LLMs interpret test instances mostly correctly in the Structure and Behavior categories. The differences in scores between the two LLMs are minor and are mostly due to the stochasticity of the answers. As the number of test instances is rather small (work in progress), a few incorrect answers can noticeably influence the final score.

In the Basic functionality category, it appears that GPT-4o outperforms GPT-4 Turbo. However, note that the scores are not equal to 1, even for correct answers (when checked manually). The differences are due to the sensitivity of the ROUGE metric (Lin, 2004) to the exact formulation of the answers. In short, the ROUGE-1 metric counts how many words from the reference answer appeared in the LLM's answer, so a correct LLM's answer formulated using different words will not necessarily get a perfect score. By manually checking the answers of both GPT-4o and GPT-4 Turbo, we noticed that both LLMs usually answer correctly, so we attribute the differences in score

[7]https://python.langchain.com/

[8]https://platform.openai.com/

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

Table 2: Summary of the evaluation results for three WADL expressiveness variants. GPT-4o results in agent mode are in black, GPT-4 Turbo in blue, and GPT-4o in up-front mode in teal (note that for the No sub-workflows variant, it does not make sense to differentiate between agent and up-front modes as there is always only one WADL specification file). The scores show the mean of five repetitions of each test instance.

Pattern category	Pattern name	Instance count	With imports WADL Score	Inlined WADL Score	No sub-workflows WADL Score
Structure	List of tasks	5	0.85 0.85 0.95	1.00 1.00 1.00	1.00 1.00
	Links in flow	4	0.95 0.95 1.00	0.95 1.00 1.00	0.95 0.85
	Task after task	4	0.75 0.60 0.95	0.85 0.75 1.00	0.70 0.60
	Next tasks in flow	4	0.95 1.00 0.80	0.90 1.00 0.88	0.78 1.00
	Flow cycle	4	0.80 0.85 1.00	1.00 1.00 1.00	0.70 1.00
	Total	*21*	0.86 0.85 0.94	0.94 0.95 0.98	0.83 0.90
Behavior	Mutually exclusive conditional flow	2	1.00 1.00 1.00	1.00 1.00 1.00	1.00 1.00
	Total	*2*	1.00 1.00 1.00	1.00 1.00 1.00	1.00 1.00
Basic functionality	Inconsistent task name and description	1	0.52 0.36 0.45	0.56 0.31 0.48	0.55 0.32
	Inconsistent descriptions	1	0.34 0.22 0.42	0.43 0.26 0.39	0.47 0.23
	Total	*2*	0.43 0.29 0.44	0.50 0.29 0.43	0.51 0.28

mainly to the GPT-4o formulating its answers closer to our reference ones. To address the issue, we experimented with the BERTScore metric (Zhang et al., 2020), which uses word embeddings to capture the meaning of words and should therefore be less sensitive to the exact wording of the answers. However, we still could not differentiate correct and incorrect answers more accurately. Therefore, we plan to conduct further research and experiment with other metrics.

We noticed that GPT-4o performs worse than GPT-4 Turbo in the No sub-workflows WADL variant in the questions that require answering only within the scope of a particular task (corresponding to a sub-workflow in other variants); for instance, this is the case in the Flow cycle pattern for which GPT-4o scored 0.7 while GPT-4 Turbo scored 1.0. We figured out that (newer) GPT-4o did not correctly interpret the instruction to work only within the scope inside the task—it incorrectly considered also the task itself. We suspect this was because we originally "fine-tuned" the instructions for GPT-4 Turbo. Altering the prompt formulation to clarify the instruction might be necessary for other LLMs (including GPT-4o).

Even though the differences between the results of the WADL expressiveness variants are minor, in the Import variant, the LLMs score worse in the Structure category by obviously tending to miss the information obtained via the importing effect (as shown in example in Figure 3), especially in the agent mode as the LLM has to explicitly ask for the WADL specification of the base workflow. In the up-front mode, the differences are not that significant.

Interestingly, presenting all the WADL specifications to the LLM up front can also hinder the results. For instance, in the Next tasks in flow pattern, the LLM in up-front mode scores worse because when the sought next task represents a sub-workflow, it looks inside it and answers with the first task from it instead of just returning the name of the task itself. In the agent mode, it would have to first ask for the WADL specification of the sub-workflow to obtain the task inside.

From these results, we conclude that our methodology is viable for evaluating the ability of LLMs to interpret WAs.

4.4 Discussion and Interesting Observations

In this section, we comment on two interesting observations based on the results.

4.4.1 Sensitivity to Prompt Formulation

As published elsewhere, it is desirable to formulate the prompt as exactly as possible. Specifically, e.g., when the question asks to "list all tasks that have a parameter", the LLMs sometimes list tasks that depend on the Hyperparameters data object. However, when the question is formulated more precisely by adding "(specified via the 'param' keyword)", we get the desired answers.

As another example, consider two patterns that target the same workflow feature but by differently

> Your goal is to help the user with analyzing results of an experiment and suggest improvements to the experiment itself. The experiment is defined by a workflow, which is an activity diagram consisting of several tasks with a data flow among them. Each of the tasks can be composed of a subworkflow and you can use tools to obtain the specification of the subworkflow.
>
> The workflow is specified using a DSL:
> arrows "->" represent control flow
> arrows with question mark "?->" represent conditional control flow
> dashed arrows "–>" represent data flow
>
> A workflow can be derived from another workflow, which is denoted by the "from" keyword. When working with a derived workflow, always obtain the specification of the base workflow as it is necessary to fully understand the derived workflow.
>
> Use the available tools if you need specification of a workflow or a task. Always gather all the necessary information before submitting your final answer. Think step by step. First, reason about the question and write a short explanation of your answer. Then, on a separate line, write "Final answer:" followed by your final answer to the question. Your final answer must be a comma separated list of values.
>
> List all tasks in workflow 'package2.MainWorkflow'. Do not list tasks nested inside other tasks.

Figure 2: Example of prompting a test instance of the List of tasks test pattern (Sect. 3.2.1) in the agent mode. In compliance with Sect. 4.2, the prompt starts with the WADL variant description, followed by the instructions to the LLM (highlighted in pink), and the question (yellow) with the workflow reference (green) inside it.

formulated questions—Links in flow asks if there "is a control flow link" between two tasks, and Task after task asks whether a task "follows directly after" another task (a more detailed description of the patterns is available in the replication package[3]). The results are better for the Links in flow pattern in all WADL expressiveness variants clearly showing that dissimilarly formulated questions about the same feature can result in different scores.

4.4.2 Semantically Incorrect Test Instances

We noticed that among the Structure test instances, performance was worse when the WADL specification in a prompt was semantically incorrect. For instance, this happened in the case of the prompt with a specification where ModelEvaluation proceeded before ModelTraining (in the control flow the evaluation should be obviously after training). When the prompt question was: "Does task 'Training' follow directly after 'Evaluation' in the control flow?", the answers were (incorrectly) negative for all WADL variants. The LLM gave illogical answers, such as: "... ModelTraining does not follow directly after ModelEvaluation ... Instead, ModelEvaluation precedes ModelTraining..."

4.5 Threats to Validity

The threats to validity are mostly related to the fact that LLMs are sensitive to the exact formulation of the questions asked.

A limitation is that the exact formulation of test patterns and test instances must be tailored to a particular ADL since ADLs may differ in the concepts they define (e.g., "sub-workflow" does not exist in all the WADL variants we tested). Thus, the difference in results between two particular ADLs may be due to specific formulations of basically similar questions and not to a different quality of interpretation of the ADLs.

Similarly, the formulation, specifically the clarity of the instructions, can influence the results of different LLMs (as we noticed in the Flow cycle pattern). It is necessary to be careful not to "fine-tune" the questions to one particular LLM but to find a trade-off that fits all the tested LLMs. Otherwise, the differences among scores of different LLMs could be due to the formulation of the questions and not to different levels of interpreting WAs by these LLMs.

Another threat is that we only used two LLMs, so the results do not refer to the performance of other LLMs available elsewhere. Nevertheless, the main goal of this paper is to provide a methodology to measure the quality of LLMs interpretation, the flexibility of our experimental framework is achieved by employing the LangChain abstractions so that swapping to another LLM requires updating only a few lines of our code.

5 RELATED WORK

The related areas of work for our paper are the evaluation of LLMs and the use of LLMs to design a software architecture.

To facilitate the evaluation of LLMs, there are a number of frameworks that allow grouping of evaluation datasets and performing a large number of evaluation tasks, such as *Language Model Evaluation Harness* (Sutawika et al., 2024) and *Holistic Evaluation of Language Models* (Liang et al., 2023). There are also several different metrics used in evaluations as summarized by (Ip, 2024), including purely statistical scores (e.g., word overlap between LLM outputs and

PROMPT TO LLM: see Figure 2

LLM ASKS FOR WORKFLOW VIA FUNCTION CALL:
workflow_specification(package='package2', workflow='MainWorkflow')

ANSWER TO LLM BY THE FUNCTION OUTPUT:

```
package package2;

workflow MainWorkflow from package1.Workflow1 {
  configure task Task3 {
    implementation "file://implementation_B.py";
  }

  configure task Task4 {
    subworkflow SubWorkflow;
  }

  configure task Task1 {
    configure tasks Task6 {
      configure task Task8 {
        implementation "file://implementation_E.exe";
      }

      configure task Task9 {
        implementation "file://implementation_F.exe";
      }
    }
  }
}
```

LLM RESPONSE:
To list all tasks in the workflow 'package2.MainWorkflow' without including tasks nested inside other tasks, we need to consider the structure of the workflow. This workflow is derived from 'package1.Workflow1', and it has been customized by adding or configuring new tasks. However, the question specifically asks not to list tasks nested inside other tasks, meaning we only focus on the top–level tasks directly under 'package2.MainWorkflow'.

From the provided workflow specification, we see that 'package2.MainWorkflow' directly configures or mentions the following tasks: Task3, Task4, and Task1. It's important to note that Task1 has nested tasks (Task6, Task8, Task9), but these are not to be listed as per the instructions.

Final answer: Task3, Task4, Task1

CORRECT ANSWER:
Task1, Task2, Task3, Task4

SCORE: 0.75

Figure 3: Example of an LLM response to the prompt in Figure 2. The LLM first requests the WADL specification of the workflow in question and then answers in the desired format. While the LLM correctly omits tasks 6, 8 and 9 which are nested inside other tasks, it does not obtain the specification of package1.Workflow1 which is imported into the workflow in question (via the from keyword). It thus misses Task 2 (defined in the imported workflow) in its final answer.

expected outputs), and model-based scores that use another ML model for their computation. An especially interesting metric is *G-Eval* (Liu et al., 2023), which uses another LLM to rate the answers of the evaluated LLM (known as the *LLM-as-a-judge* approach). An interesting approach to evaluating LLMs is *Chatbot Arena* (Zheng et al., 2023). It is a crowd-sourcing platform where users prompt questions to two anonymous LLMs and are then asked to pick the LLM with the best response. LLMs are rated using the Elo rating system based on the results of the "duel".

The data sets mentioned above often focus on a broad evaluation of LLMs on a wide range of tasks. There are also data sets focused on tasks related to software engineering, such as *DevBench* (Li et al., 2024) focusing on software development (including software design, implementation, and testing). Nevertheless, as LLMs have gained widespread usage only very recently, there have been not many works on employing LLMs during software architecture design. (Ahmad et al., 2023) show a case study of using LLM (ChatGPT in particular) during the design of software architecture. The paper presents a process of interaction of a software architect with LLM, but any actual evaluation of the process, the effectiveness of LLM, etc. is left to future work. (Dhar et al., 2024) analyze the effectiveness of LLMs in generating architectural design decisions. They give an LLM a context of the required decision and ask to make a decision. They compare several LLMs and approaches to asking them and conclude that LLM generates design decisions but does not attain human-level correctness. The context they give to the LLM is human-written requirements, while we use WADL to capture a WA.

Several works are focused on evaluating how well LLMs understand graphs (Fatemi et al., 2023; Wang et al., 2024; Guo et al., 2023). Even though they do not target software architectures, they are relevant by evaluating encodings of non-textual data for LLMs. They employ a similar approach to ours—creating a set of test instance questions and testing how well an LLM can answer the questions with different encodings of the graph structure (a graph encoding usually represents the nodes by integers and explicitly lists the edges). They also experimented with different encodings, such as giving nodes English names and edges represented by "friendship". In contrast, we use an external DSL (WADL) to represent workflow architectures.

6 CONCLUSION

In the paper, we have outlined a methodology for testing the quality of interpretation of workflow architectures by LLMs. Stemming from the conjecture that LLM should correctly answer low-abstraction-level questions to respond to those at a higher level of abstraction reliably, it introduces a set of test patterns, intended to generate a series of low-abstraction queries testing the reliability of LLM answers. Although the presented list of test patterns is not exhaustive, the initial results indicate that this approach is viable.

The main lessons learned from the results were as follows.

- LLMs can reasonably interpret workflow architectures to answer questions about their structure, behavior, and basic functionality.
- The answers of LLMs are subject to aleatoric uncertainty—the LLM can give different results to the same question. However, taking the majority vote (of 5 repetitions in our case) gives a correct answer to almost all of our test instances (22–24 correct out of 25 instances depending on the LLM and WADL variant).
- The “problematic” test instances differ among the LLMs and WADL variants. The answers tend to be worse when the workflow is semantically incorrect (Sect. 4.4.2), and in the case of Import WADL variant (only in the agent mode, e.g., List of tasks pattern in Table 2).
- It is necessary to formulate the questions as clearly and accurately as possible (Sect. 4.4.1).
- Instructing the LLM to reason about the question before answering it improves the results. (Kojima et al., 2023)
- The ROUGE (Lin, 2004) and BERTScore (Zhang et al., 2020) metrics are not good enough to evaluate open-ended questions.

In the future, we plan to extend the methodology by instantiating more test patterns and by identifying a better evaluation metric for the Basic functionality category, and apply it to questions at a higher abstraction level, such as recommending a task fitting into the given workflow architecture.

ACKNOWLEDGEMENTS

This work was partially supported by the EU project ExtremeXP grant agreement 101093164, partially by INTER-EUREKA project LUE231027, partially by Charles University institutional funding 260698, and partially by the Charles University Grant Agency project 269723.

REFERENCES

Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Aktar, M. S., and Mikkonen, T. (2023). Towards Human-Bot Collaborative Software Architecting with ChatGPT. In *Proceedings of EASE 2023, Oulu, Finland*, pages 279–285. ACM.

Dhar, R., Vaidhyanathan, K., and Varma, V. (2024). Can LLMs generate architectural design decisions? - An exploratory empirical study. In *Proceedings of ICSA 2024, Hyderabad, India*, pages 79–89. IEEE CS.

Fatemi, B., Halcrow, J., and Perozzi, B. (2023). Talk like a graph: Encoding graphs for large language models.

Guo, J., Du, L., Liu, H., Zhou, M., He, X., and Han, S. (2023). Gpt4graph: Can large language models understand graph structured data? An empirical evaluation and benchmarking.

Ip, J. (2024). LLM evaluation metrics: Everything you need for LLM evaluation.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2023). Large language models are zero-shot reasoners.

Li, B., Wu, W., Tang, Z., Shi, L., Yang, J., Li, J., Yao, S., Qian, C., Hui, B., Zhang, Q., Yu, Z., Du, H., Yang, P., Lin, D., Peng, C., and Chen, K. (2024). DevBench: A comprehensive benchmark for software development.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., Newman, B., Yuan, B., Yan, B., Zhang, C., Cosgrove, C., Manning, C. D., Ré, C., Acosta-Navas, D., Hudson, D. A., ..., and Koreeda, Y. (2023). Holistic evaluation of language models.

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. (2023). G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of EMNLP 2023, Singapore*.

Sutawika, L., Schoelkopf, H., Gao, L., Abbasi, B., Biderman, S., Tow, J., ben fattori, Lovering, C., farzanehnakhaee70, Phang, J., Thite, A., Fazz, Wang, T., Muennighoff, N., Aflah, sdtblck, nopperl, gakada, tttyuntian, ..., and AndyZwei (2024). Eleutherai/lm-evaluation-harness: v0.4.2.

Wang, H., Feng, S., He, T., Tan, Z., Han, X., and Tsvetkov, Y. (2024). Can language models solve graph problems in natural language?

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020). BERTScore: Evaluating text generation with BERT.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. (2023). Judging LLM-as-a-judge with MT-Bench and Chatbot Arena.