

Can Large Language Models Engineer Parallel Workflows?

Paula Woyames¹, Liliane Kunstmann², Débora Pina³, Marta Mattoso⁴, Daniel de Oliveira¹

¹Institute of Computing – Universidade Federal Fluminense (UFF)

²Department of Computing – Universidade Federal Rural do Rio de Janeiro (UFRRJ)

³Computer Sciences Department – Barcelona Supercomputing Center (BSC)

⁴COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

pnwoyames@id.uff.br, debora.pina@bsc.es

liliane.kunstmann@ufrrj.br, marta@cos.ufrj.br, danielcmo@ic.uff.br

Abstract. *Scientific workflows are specified using specialized languages to define parallel execution, an expertise that domain users may lack. Large Language Models (LLMs) can lower this barrier, but existing evaluations stop at syntactic analysis and do not verify whether the generated specifications execute or preserve parallelism. In this paper, we evaluated four LLMs in two workflow engineering tasks (workflow specification from natural-language and conversion across workflow systems), covering three workflow systems, three workflows, and two prompting strategies. For each generated workflow specification, we analyzed its executability and whether it executes in parallel. Results show that lexical similarity to expert-developed workflows is a poor predictor of executability, that in-context examples improve execution rates unevenly across systems, and that several workflow specifications collapse into serial execution. Even those that support parallelism are consistently slower than the expert-developed workflow.*

1. Introduction

Many scientific applications rely on the *Scientific Workflow* [de Oliveira et al. 2019] abstraction (hereafter simply referred to as *workflow*) to model complex flow of tasks that includes data preparation, simulation, analysis, and post-processing. Workflow systems automate workflow execution while handling task scheduling, provenance collection, and fault tolerance [Liu et al. 2015]. Due to the large volume of data to be processed, many workflows are routinely executed on distributed infrastructures ranging from High Performance Computing (HPC) clusters to cloud platforms [Badia et al. 2017]. Although several workflow systems support these environments, they vary in how workflows are specified and parallelized.

Despite initiatives toward standardization, *e.g.*, Common Workflow Language (CWL) [Amstutz et al. 2016], no workflow specification language has emerged as a universal standard. Instead, each system adopts its own programming model, syntax, and parallel strategy. For example, Parsl [Babuji et al. 2019] and PyCOMPSs [Badia et al. 2026] specify workflows as Python programs with decorators (that define the part of the code will execute in parallel), Nextflow [Di Tommaso et al. 2017] employs a Groovy-based domain-specific language (it executes one independent task per set of data items available in a process’s input channels), whereas Pegasus [Deelman et al. 2015] and SciCumulus [de Oliveira et al. 2010] rely on XML-based languages (where parallelism comes from an explicitly declaration, not from inferred dataflow). As a result, workflow specifications are often tightly coupled to their

underlying execution systems. This heterogeneity creates an obstacle for workflow development, reuse, and portability [Da Silva et al. 2024]. Domain experts therefore often depend on workflow and HPC experts to implement and adapt workflows across systems.

Recent advances in Large Language Models (LLMs) [Zhao et al. 2023] have shown capabilities in code generation [Chen et al. 2021]. These advances suggest that LLMs could lower the barrier to workflow engineering by enabling users to generate workflow specifications directly from natural-language descriptions and to reuse existing workflows by converting them across different workflow systems. However, applying LLMs to workflow engineering presents challenges distinct from those of conventional code generation. A workflow specification must not only be syntactically valid but also correctly represent the dataflow and opportunities for parallel execution, *e.g.*, directives that transform functions into concurrent tasks. A workflow that is functionally correct but fails to provide these execution semantics may degrade performance.

Existing papers investigating LLMs for workflow specification remain preliminary [Yildiz and Peterka 2025, Yin et al. 2025, Miyashita et al. 2024]. To the best of the authors’ knowledge, prior work either does not investigate multiple workflow engineering tasks, does not systematically compare prompt engineering strategies, or does not evaluate whether generated workflows preserve execution performance in real multi-core/distributed environments. Thus, it remains unclear whether current LLMs are suitable for practical parallel workflow engineering beyond the production of syntactically valid specifications.

In this paper, we present a study of LLMs for engineering workflows. We investigate two fundamental engineering tasks: *(i)* workflow specification, in which workflow definitions are generated from natural-language requirements, and *(ii)* workflow conversion, in which workflows are translated across different specification languages and workflow systems. Beyond evaluating syntactic and semantic correctness, we evaluate whether the generated workflows explore parallelism. We further compare multiple prompt engineering strategies to understand how prompting affects generation quality across different workflow specifications.

2. Related Work

Closest to our study, [Yildiz and Peterka 2025] evaluate whether LLMs can produce specifications for workflow systems, and [Yin et al. 2025, Miyashita et al. 2024] adapt LLMs to HPC users through retrieval-augmented and domain-specialized assistants. These papers establish that LLMs can generate plausible workflow code, but they stop at generation. None of them executes the produced specifications, compares prompting strategies under a controlled factorial design, or asks whether the resulting workflow still executes its tasks concurrently. Our study differs on exactly these three points.

[Duque et al. 2023] argue that LLMs allow workflows to be specified using natural-language, and propose a workflow engine that translates textual descriptions into user functions via OpenAI’s function-calling API. [Sänger et al. 2024] evaluate GPT-3.5 for workflow comprehension, modification, and extension through three qualitative user studies covering five real-world Nextflow and Apache Airflow pipelines. A script is considered recoverable if a domain expert repairs it within a fixed 20-minute window, a pass/fail call made by the same person performing the repair, without independent confirmation. [Ljaljevic et al. 2026] survey 32 studies on LLMs for HPC programming. Two findings are directly relevant in the context of this paper: *(i)* generic similarity scores such as BLEU correlate only weakly with program semantics or runtime behavior, and *(ii)* the field lacks evaluation frameworks that jointly

capture executability and functional correctness. Our design addresses this gap by reporting BLEU and chrF alongside an independent execution-based success rate, keeping code similarity and functional correctness separate rather than conflating them. [Alam and Roy 2025] evaluate GPT-4o, Gemini 2.5 Flash, and DeepSeek-V3 on generating Galaxy and Nextflow workflows, scoring outputs against workflow baselines through manual review by two domain experts. [Gu et al. 2023] propose PGM (*Plan, Generate and Match*), a three-stage workflow recommendation method in which an LLM plans, generates a structured specification, and matches candidate workflows to the user query. PGM does not evaluate the efficiency of the recommended specification.

3. Methodology

This section describes the methodology adopted to evaluate the engineering of parallel workflows. The methodology is organized around two research questions, each corresponding to an engineering task. The first question concerns the *Specification* of a parallel workflow from a natural-language requirement, while the second concerns the *Conversion* of an existing parallel workflow specification across workflow systems. The first research question (RQ1) therefore evaluates whether LLMs can reduce this barrier by producing specifications that are not only syntactically correct, but structurally comparable to expert-developed reference parallel implementations. Thus, RQ1 can be defined as “*To what extent can LLMs generate parallel workflow specifications from natural-language requirements that are syntactically valid, executable, and semantically faithful to the intended dataflow?*”.

Converting a parallel workflow between systems currently requires expertise in both the source and the target specification languages, and can be an obstacle to workflow reuse. RQ2 evaluates whether LLMs can act as a translation layer that relieves the user from learning a second system, and whether the converted specifications preserve the semantics of the original one, including parallel capabilities. Thus, RQ2 can be defined as “*To what extent can LLMs convert parallel workflow specifications across workflow systems while preserving tasks, dataflow dependencies, and parallel capabilities?*”.

We evaluate four LLMs: (i) Command-A-Plus-05-2026 [Cohere 2026], (ii) Gemma-4-31B-IT [Google 2026], (iii) GPT-5.5 [OpenAI 2026], and (iv) DeepSeek-V4-Flash [Deepseek 2026]. For each LLM, we use the most recent checkpoint available at the time of the experiments. The models were selected before any results were inspected, so that the evaluation is the focus of this study and not an input to it. Table 1 summarizes the chosen models and their configurations. These four models span two deployment paradigms (proprietary and open-weight) and different parameter scales, allowing us to evaluate whether the observed behavior is consistent across models. We use the term *open-weight* rather than *open-source* for the three non-proprietary models, since their parameters are public but their training corpora are not.

Table 1. LLMs evaluated in this study.

Model	Provider	Release	Params	Access
GPT-5.5	OpenAI	2026-04-23	n/d	P
DeepSeek-V4-Flash	DeepSeek	2026-04-24	13B	OW
Command-A-Plus	Cohere	2026-05-20	25B	OW
Gemma-4-31B-IT	Google	2026-04-02	30.7B	OW

We consider three workflow systems that support workflow parallelism and differ

in their parallelization models and specification languages: (i) Parsl, (ii) PyCOMPSs, and (iii) Nextflow. This heterogeneity is intentional to evaluate whether the observed results are consistent across systems or instead concentrated in systems whose languages are more likely to be overrepresented in the models’ training data. For example, Parsl and PyCOMPSs are based on Python, which is likely well represented in the corpora used to train the LLMs, whereas Nextflow uses a Groovy-based DSL that is likely less prevalent in those corpora.

Each prompt submitted to the LLMs requests three workflows, all parallel ones, chosen to span the spectrum from pure computation to real scientific simulation, thereby progressively exercising more demanding aspects of workflow specification: (i) K-Means is a clustering workflow that explores the ability to express iterative, data-parallel task decomposition and reduction over partitioned input, (ii) MatMul is a compute-intensive blocked matrix multiplication workflow, and it exercises the expression of fine-grained task parallelism and of dependencies induced by block partitioning, and (iii) Lysozyme is a molecular dynamics workflow executed with GROMACS. It contains external calls to binaries (`gmx` and `gmx_mpi`). For each workflow/workflow-system pair, we specified a *ground truth* with assistance from workflow experts and validated it in the target system. These specifications provide the reference against which structural and lexical similarity are evaluated and also form the basis for the examples presented under the few-shot prompts.

According to [Sahoo et al. 2024], prompt engineering steers model outputs by strategically constructing the input without modifying model parameters. In this paper, we investigate two prompt strategies while keeping the task description fixed and varying only the presence of examples: (i) Zero-shot, in which the model receives the task description together with the minimal set of application-specific constraints, but no example written in the target specification language; and (ii) Few-shot, in which the zero-shot prompt is augmented with one reference specification written for the target workflow system. The reference specification presents characteristic constructs of the target language, including annotations, dependency declarations, and execution directives. This design isolates the contribution of examples to the quality of generation. In particular, it allows us to evaluate whether providing workflow examples can compensate for the under-representation of a given workflow specification language in the model’s pre-training data. The following prompts P1–P4, reproduce the templates used in our experiments. Placeholders enclosed in [brackets] are instantiated according to the experimental condition, while `<target system>` is replaced by Parsl, PyCOMPSs, or Nextflow. PyCOMPSs is used as the fixed source system in RQ2, so that the conversion task is evaluated over a common source.

P1 — RQ1, Zero-shot

Write a workflow specification using `<target system>` to generate parallel workflows that compute the K-Means, MatMul, and Lysozyme applications separately. The Lysozyme application requires `gmx` and `gmx_mpi` from GROMACS, with topology generation performed by `pdb2gmx`.

P2 — RQ1, Few-shot extends P1 with: *Below is an example of a workflow in the target system, summarizing some of its constructs. Reference script:* [reference specification]

P3 — RQ2, Zero-shot

Translate the following PyCOMPSs workflows (K-Means, MatMul, and Lysozyme, respectively) to `<target system>`. Scripts to translate: [source specifications]

P4 — RQ2, Few-shot extends P3 with: *After the input scripts, a reference workflow written in the target system is provided to guide the construction of the answer.* [reference

Listing 1. Repair ρ turning an ER failure into an ER⁺ success (Gemma-4-31B-IT, zero-shot, Parsl K-Means).

```
- from parsl.app.app import python_app, shell_app
+ from parsl.app.app import python_app
- from parsl.configs.local_cluster import config as local_config
+ from parsl.configs.local_threads import config as local_config
```

specifications in the target system]

Each of the aforementioned prompts is submitted in an independent LLM session, with no conversational history or augmentation, so cross-condition contamination is avoided. Parameters are held constant across all models in rounds 2–5, submitted through APIs (temperature = 0.7 with no seed fixed, as not all providers expose one); round 1 used each web interface’s default settings. No output-token limit was imposed. No response approached the models’ output limits, so no truncation occurred. Additionally, we relied on the default settings for any models that offered reasoning-effort or thinking-budget controls, rather than defining those parameters ourselves. We extract the model output directly from its response, removing only surrounding prose and markdowns. Each generated workflow specification is subsequently deployed and executed to completion in the target system, whose hardware and software configuration is reported in Section 4.1. Generated specifications and their corresponding reference specifications use identical input datasets and problem sizes. Thus, both workflows are executed under equivalent computational conditions, allowing differences in execution behavior to be attributed to the generated specifications rather than to variations in workload or input data. To support reproducibility, all prompts, workflow specifications, and scripts are publicly available at <http://github.com/UFFeScience/sscad-2026>.

We evaluate the workflow specifications generate to answer RQ1 and RQ2 along three dimensions: (i) executability, (ii) lexical similarity, and (iii) parallel capabilities. Let $\mathcal{S}$ denote the set of workflow specifications produced under a given experimental configuration. Every generated or translated workflow specification is executed until completion. Regarding executability, a specification is recorded as successful when it terminates without errors, and as failed otherwise, including cases in which only minor adjustments would be required. The execution rate can be defined as $ER(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \mathbb{I}[\text{exec}(s) = \text{success}]$, where $\text{exec}(s)$ denotes execution of a given workflow specification in the target system and $\mathbb{I}[\cdot]$ is the indicator function. As a secondary metric, we measure the success rate obtained after applying a restricted repair procedure ρ . The procedure admits only surface-level edits, *e.g.*, missing imports, and is forbidden from altering the tasks, the dependency structure, or the control logic of the specification. Formally, $ER^+(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \mathbb{I}[\text{exec}(\rho(s)) = \text{success}]$, where $ER^+ \geq ER$. The gap $ER^+ - ER$ separates the cases in which the model produced a specification very close to correct but with superficial defects from those in which the generated specification is non-executable at a deeper structural level. The criteria distinguishing an admissible surface-level edit from an inadmissible structural one were fixed before any generated specification was inspected, and both the original and the repaired workflows are published in the repository so that this classification can be audited independently. As seen in Listing 1’s representative application of ρ from the published artifacts, two imports for absent Parsl modules are updated while the tasks and dependencies stay the same.

Because an error-free execution is not synonymous with correctness, we evaluate K-Means and MatMul outputs using a correctness oracle and a non-intrusive profiler during

re-execution. The MatMul operation must satisfy $\|C - AB\|_F \leq 10^{-6} \|A\|_F \|B\|_F$, and the final K-Means centroids must either be a valid Lloyd update (relative error $\leq 10^{-6}$) or a converged solution with an inertia decrease of at most 10^{-4} . The oracles approved all expert-created references and flagged six incorrect versions that did not crash. Of the 248 executable K-Means and MatMul specifications, 25 were marked as failures in ER and ER⁺, including 14 non-functional placeholders, 8 algorithmic flaws, and 3 unexecutable generated files, with all final verdicts being made available.

Regarding lexical similarity, we employ two similarity metrics, chrF [Popović 2015] and BLEU [Papineni et al. 2002], to compare the generated specifications against the ground truth. BLEU measures similarity through the overlap of token n -grams between the answer and the ground truth, capturing correspondence at the lexical and structural levels. chrF, in contrast, operates over character n -grams, which makes it more robust to minor variations in formatting. We compute both metrics with SacreBLEU¹, setting $\beta = 2$ and $N = 6$ for chrF. For BLEU, we disable SacreBLEU’s built-in tokenizer (`tokenize=none`) and supply candidates and references pre-tokenized by a code-aware tokenizer, since standard natural-language tokenization schemes are not appropriate for source code. Both are reported on a $[0, 100]$ scale (higher is more similar). We stress that lexical similarity is a proxy for, rather than a criterion of, correctness; *i.e.*, a workflow specification may be semantically equivalent to its reference while diverging substantially in syntax, and conversely, a highly similar specification may still fail to execute.

Finally, since executability and similarity do not imply that a workflow runs in parallel, every executable workflow is run in a real environment, collecting execution traces to observe the parallelism actually achieved and the makespan.

The corresponding experiments follow a factorial design over $4 \text{ LLMs} \times 2 \text{ engineering tasks (workflow specification/conversion)} \times 2 \text{ prompt strategies}$, crossed with 3 workflow systems for the *Specification* task and 2 workflow systems (Parsl and Nextflow) for the *Conversion* task (since PyCOMPSs is used as the source system being translated from and is therefore never itself a target). This yields $4 \times 3 \times 2 + 4 \times 2 \times 2 = 40$ prompt instances per round, each requesting three workflows (120 specifications); with five rounds, the design comprises 600 workflow specifications.

4. Experimental Results

This section reports the results of the empirical study according to the three dimensions (executability, lexical similarity, and parallel capabilities) defined in Section 3.

4.1. Environment Setup

All workflow specifications were generated or translated on a machine equipped with an Apple M5 chip (10-core CPU, 10-core GPU) and 24 GB RAM. Each prompt was submitted five times to each LLM. Repeating each prompt helps attenuate the stochastic variability in the responses. All similarity values are reported as mean $\pm$ standard deviation over the five independent samples. The pipeline was automated with scripts available in the repository. Round 1 was collected through each provider’s web interface and in rounds 2 to 5, a driver submitted the prompts through the OpenRouter gateway² (GPT-5.5, DeepSeek-V4-Flash, Gemma-4-31B-IT) and Cohere’s API (Command-A-Plus, unavailable on OpenRouter). Code was

¹<https://github.com/mjpost/sacrebleu>

²<https://openrouter.ai/>

then extracted per application and executed in isolated sandboxes (Docker for PyCOMPSs). Failed specifications were repaired under ρ on isolated copies and re-executed. Every repair was logged and reviewed by the authors.

Workflow generation and execution were carried out in different environments. Every generated specification, with its expert-developed reference, was deployed and executed on a single Amazon AWS `c5.4xlarge` VM with 16 vCPUs (Intel Xeon Platinum 8124M) and 32 GiB of RAM, running Ubuntu 22.04 LTS. The workflow systems and supporting software were used at the following versions: Parsl 1.3.0-dev, PyCOMPSs version 3.4, Nextflow v26.07.0-edge, and Python 3.14.7. Generated and reference specifications were given identical inputs and problem sizes, and each workflow was executed five times so that the makespan reflects an average behavior rather than a single execution.

4.2. Evaluation of Workflow Specification Task

To answer RQ1, we used the `P1–P4` prompts to ask the four LLMs to generate workflow specifications for Nextflow, Parsl, and PyCOMPSs from natural-language descriptions. For each prompting strategy, each LLM generated three specifications for each workflow system. This resulted in 72 specifications per repetition and 360 specifications across the five repetitions. Table 2 reports both prompting strategies side by side. Considering the zero-shot columns first, across all three workflow systems, GPT-5.5 produced specifications that were structurally closest to the ground truth and achieved the highest execution rates, both before and after repair. Although the other LLMs sometimes achieved similarity scores close to GPT-5.5’s, they generally failed to produce executable specifications, with a success rate of zero across nearly all configurations.

This contrast is the first finding of this paper: high structural similarity does not necessarily imply functional correctness. A LLM can generate a specification that is lexically close to the ground truth while still failing to execute. An example is Command-A-Plus on Parsl. Its BLEU score of 11.54 is comparable to GPT-5.5’s score of 11.69, yet its execution rate is 0.0% both before and after repair. As in the other experiments, Nextflow yielded the lowest similarity scores. We attribute this result to its more limited adoption and, consequently, to the smaller amount of documentation likely available in the models’ pre-training data. However, lower similarity did not reduce executability for the strongest model. GPT-5.5 achieved its highest execution rate on Nextflow (80.0%). This result suggests that the Groovy-based DSL may make the generated specifications less similar to a specific reference while still allowing the model to produce valid and executable workflows.

Turning to the ER^+ metric, we observe improvements for nearly all workflow systems and models, with the exception of Command-A-Plus. In most cases, the required repairs were minor, consisting mainly of correcting parameter declarations and malformed syntax rather than changing the overall workflow structure. This suggests that several LLM-generated specifications were nearly executable but contained errors in key configuration directives. The improvement is particularly clear for Gemma-4-31B-IT. Its execution rate increases from 0.0% to 53.3% on both Parsl and PyCOMPSs. Thus, specifications that initially appeared to be complete failures could, in more than half of the cases, be made executable through relatively simple repairs. Throughout the experiment, we found that most errors were structural rather than semantic. Common problems included incorrect module imports, misconfigured settings, references to non-existent modules, and calls to APIs that are not part of the target workflow system.

Table 2. RQ1 (specification): zero-shot (ZS) and few-shot (FS) prompts.

Model	BLEU $\uparrow$		chrF $\uparrow$		ER (%) $\uparrow$		ER ⁺ (%) $\uparrow$	
	ZS	FS	ZS	FS	ZS	FS	ZS	FS
Parsl								
GPT-5.5	11.69 ± 3.07	11.98 ± 2.43	33.32 ± 4.56	36.22 ± 4.50	60.0	60.0	93.3	93.3
DeepSeek-V4	7.24 ± 0.73	7.79 ± 1.62	25.97 ± 1.15	24.29 ± 2.72	13.3	53.3	93.3	80.0
Command-A-Plus	11.54 ± 2.42	9.70 ± 2.00	28.97 ± 1.19	28.10 ± 1.27	0.0	26.7	0.0	46.7
Gemma-4-31B-IT	4.19 ± 0.80	3.08 ± 0.25	19.69 ± 0.70	18.49 ± 0.68	0.0	46.7	53.3	46.7
<i>Average</i>	<i>8.67</i>	<i>8.13</i>	<i>26.99</i>	<i>26.77</i>	<i>18.3</i>	<i>46.7</i>	<i>60.0</i>	<i>66.7</i>
PyCOMPSs								
GPT-5.5	15.85 ± 2.26	14.99 ± 3.08	34.03 ± 3.82	37.31 ± 3.94	40.0	53.3	80.0	80.0
DeepSeek-V4	10.05 ± 2.64	14.22 ± 0.49	25.56 ± 2.21	35.84 ± 1.27	0.0	13.3	53.3	40.0
Command-A-Plus	13.48 ± 0.45	12.93 ± 2.06	26.09 ± 1.62	31.12 ± 0.95	0.0	20.0	0.0	26.7
Gemma-4-31B-IT	2.11 ± 0.89	4.84 ± 0.64	12.71 ± 0.84	20.08 ± 1.21	0.0	26.7	53.3	66.7
<i>Average</i>	<i>10.37</i>	<i>11.69</i>	<i>24.60</i>	<i>30.92</i>	<i>10.0</i>	<i>28.3</i>	<i>46.7</i>	<i>53.3</i>
Nextflow								
GPT-5.5	10.41 ± 2.69	11.30 ± 1.24	26.58 ± 4.56	30.06 ± 3.23	80.0	13.3	80.0	13.3
DeepSeek-V4	4.62 ± 1.77	4.28 ± 1.28	19.86 ± 2.17	18.90 ± 2.06	0.0	20.0	40.0	46.7
Command-A-Plus	5.13 ± 0.30	4.59 ± 0.82	22.86 ± 1.56	19.03 ± 1.08	0.0	0.0	0.0	33.3
Gemma-4-31B-IT	0.63 ± 0.13	0.94 ± 0.16	12.58 ± 0.35	13.39 ± 0.29	13.3	0.0	13.3	6.7
<i>Average</i>	<i>5.20</i>	<i>5.35</i>	<i>20.47</i>	<i>20.48</i>	<i>23.3</i>	<i>8.3</i>	<i>33.3</i>	<i>25.0</i>

Turning to the few-shot columns of Table 2, the inclusion of examples leads to improvements in execution rates. The average ER increases from 18.3% to 46.7% on Parsl and from 10.0% to 28.3% on PyCOMPSs. Across all three workflow systems, GPT-5.5 again achieved the highest structural-similarity scores. Gemma-4-31B-IT shows why executability alone is insufficient: its few-shot rates on Parsl (66.7%) and Nextflow (26.7%) drop to 46.7% and 0.0% once placeholders, which only print a message or report random numbers, are discarded by the oracles. Command-A-Plus also improved compared with the zero-shot. While it previously failed to produce executable specifications for any workflow system, it generated executable specifications for Parsl and PyCOMPSs under the few-shot, remaining unsuccessful only on Nextflow.

The ER⁺ metric again shows higher execution rates after repair, with GPT-5.5 exceeding its own unrepaired results and achieving the best overall values on Parsl and PyCOMPSs. Moreover, every workflow system-model combination produced at least one executable specification, whereas this did not occur under zero-shot prompting. Overall, providing examples improved the executability of the generated specifications for most workflow systems. The remaining failures were mainly associated with external calls that none of the models could correctly invoke and configure, consistent with the Lysozyme coverage reported in Section 4.4. Nextflow is the main exception, since its average ER decreases from 23.3% under zero-shot prompting to 8.3% under few-shot prompting. Besides the placeholders, part of the decrease is an artifact of our execution procedure rather than of the generated code. In several cases, the model wrapped each workflow in a named sub-workflow to be selected with `-entry`, a valid Nextflow DSL2 construct that our procedure does not accept. We attribute this to the prompt’s request for three independent, self-contained files, which the LLM satisfies on Nextflow by mirroring the argument-selectable organization it uses for Parsl and PyCOMPSs. These files remain valid Nextflow code, so this figure understates few-shot performance on Nextflow.

Takeaway (RQ1). LLMs can generate workflow specifications that are syntactically valid but are not consistently executable without intervention. The highest average unrepaired execution rate across all configurations is 46.7%. Providing in-context examples improves executability, particularly for the Python-based workflow systems. Repairs also recover a large fraction of the remaining failures, suggesting that the main challenge is not expressing the intended dataflow, but correctly reproducing the required configuration and API usage.

4.3. Evaluation of Workflow Conversion Task

To answer RQ2, we asked the LLMs to convert workflows from the fixed source workflow system, *i.e.*, PyCOMPSs, to two target systems, Parsl and Nextflow. As in the previous experiment, we considered zero-shot and few-shot. In both prompts, the PyCOMPSs specifications are provided as input. Under the few-shot strategy, the prompt also includes a reference workflow written in the target system. Each LLM generated three workflow specifications for each target system under each prompt strategy. This resulted in 240 specifications across the five repetitions. Table 3 reports both prompting for the conversion task. Under zero-shot, structural similarity scores are higher than those in the workflow specification experiment.

For Parsl, the average BLEU score reaches 49.48, compared with 8.67 for the specification experiment, while the average chrF score reaches 68.70, compared with 26.99. The source specification already fixes the workflow structure and dataflow, giving the model a strong reference. For Parsl, DeepSeek-V4 achieved the highest agreement with the ground truth and the highest repaired execution rate, reaching 93.3%. GPT-5.5, however, achieved the highest unrepaired execution rate at 66.7%. For Nextflow, the models struggled both to generate executable specifications and to produce outputs that closely matched the ground truth. The average ER was only 1.7%, and GPT-5.5 was the only model to produce any executable specifications. More importantly, even GPT-5.5 and DeepSeek-V4, which achieved the highest structural-similarity scores for Nextflow, produced specifications with low executability.

The difference between ER and ER⁺ is also informative. On Parsl, the average execution rate increases from 35.0% to 65.0% after repair, indicating that many conversions were functionally correct but failed because of relatively minor issues. Nextflow presents a different challenge. The models often reproduced elements of the PyCOMPSs structure instead of adapting the workflow to the target language. *i.e.*, Parsl and pyCOMPSs share a task-based model, so conversion preserves the parallel structure, whereas Nextflow encodes parallelism in channel cardinality rather than control flow. DeepSeek-V4, in particular, omitted the configuration calls required at the beginning of the specification, which halted execution. This failure type is specific to the conversion task but is related to the errors observed in RQ1. Under few-shot prompting, on Parsl, DeepSeek-V4 again achieved the highest agreement with the ground truth and the highest repaired execution rate, while Gemma-4-31B-IT achieved the highest unrepaired execution rate at 60.0%. On Nextflow, GPT-5.5 achieved the best results across all evaluation metrics. On average, the few-shot strategy improved execution rates compared with the zero-shot strategy, increasing from 35.0% to 45.0% on Parsl and from 1.7% to 3.3% on Nextflow. Despite this improvement, the results show that LLMs still have difficulty adapting PyCOMPSs specifications to Nextflow.

The repaired execution rate also improved compared with the zero-shot prompt. The largest improvement was observed for GPT-5.5 on Nextflow, where the execution rate increased from 53.3% to 86.7% after repair. The remaining failures were mostly due to syntax

Table 3. RQ2 (conversion): zero-shot (ZS) and few-shot (FS) prompts.

Model	BLEU $\uparrow$		chrF $\uparrow$		ER (%) $\uparrow$		ER ⁺ (%) $\uparrow$	
	ZS	FS	ZS	FS	ZS	FS	ZS	FS
Parsl								
GPT-5.5	48.08 $\pm$ 5.41	53.64 $\pm$ 7.14	68.73 $\pm$ 6.22	71.80 $\pm$ 7.21	66.7	53.3	66.7	80.0
DeepSeek-V4	65.95 $\pm$ 2.31	58.85 $\pm$ 3.02	80.68 $\pm$ 1.41	78.05 $\pm$ 1.12	53.3	40.0	93.3	93.3
Command-A-Plus	35.65 $\pm$ 3.57	38.98 $\pm$ 3.05	58.52 $\pm$ 3.81	67.11 $\pm$ 1.07	6.7	26.7	13.3	60.0
Gemma-4-31B-IT	48.26 $\pm$ 2.20	51.26 $\pm$ 7.06	66.86 $\pm$ 1.93	69.02 $\pm$ 5.93	13.3	60.0	86.7	60.0
<i>Average</i>	<i>49.48</i>	<i>50.68</i>	<i>68.70</i>	<i>71.49</i>	<i>35.0</i>	<i>45.0</i>	<i>65.0</i>	<i>73.3</i>
Nextflow								
GPT-5.5	32.30 $\pm$ 2.23	28.71 $\pm$ 2.96	51.26 $\pm$ 3.86	50.93 $\pm$ 4.90	6.7	13.3	53.3	86.7
DeepSeek-V4	23.33 $\pm$ 1.33	24.04 $\pm$ 1.09	45.68 $\pm$ 1.25	44.68 $\pm$ 1.28	0.0	0.0	6.7	6.7
Command-A-Plus	19.42 $\pm$ 0.90	13.35 $\pm$ 1.90	44.80 $\pm$ 0.71	38.01 $\pm$ 2.91	0.0	0.0	0.0	0.0
Gemma-4-31B-IT	8.69 $\pm$ 1.09	27.62 $\pm$ 0.33	27.58 $\pm$ 1.21	43.17 $\pm$ 0.48	0.0	0.0	0.0	20.0
<i>Average</i>	<i>20.93</i>	<i>23.43</i>	<i>42.33</i>	<i>44.20</i>	<i>1.7</i>	<i>3.3</i>	<i>15.0</i>	<i>28.3</i>

errors and references to elements not defined in the specification, as observed in Command-A-Plus. The gap between ER and ER⁺ is also much larger for workflow conversion than for workflow specification. This result suggests that when the source specification is provided, the model eventually preserves much of the original workflow structure and dataflow. The main remaining challenge is then to express this structure correctly in the target system, particularly through target-specific configuration and API usage. These are precisely the types of errors that ρ repairs.

Takeaway (RQ2). Workflow conversion produces higher lexical agreement with the ground truth than direct specification, but this similarity does not translate into executability on Nextflow, where the average unrepaired execution rate remains below 5%. The main failure mode is the transfer of source-system structures to a target system with different execution semantics. Because BLEU and chrF measure textual similarity, they cannot detect whether a structurally similar specification is valid or executable in the target system.

4.4. Evaluation of Parallelism

Executability shows that a workflow specification can execute, but it does not show that it executes exploiting *parallelism*, benefiting from the HPC or cloud environment. A workflow specification may execute successfully while scheduling all tasks sequentially, thereby losing the parallelism that motivates the use of the aforementioned workflow systems. We therefore analyze whether the LLM-generated workflows preserve the workflow’s parallel structure. Of the 120 model $\times$ prompt $\times$ RQ $\times$ workflow-system $\times$ workflow combinations, only 91 (78%) yielded a parallel/serial analysis, since the remainder could not be reliably measured. Parallel execution is detected in all K-Means and MatMul specifications, on all three workflow systems, whereas Lysozyme ranges from 50% on Parsl down to 14% on PyCOMPSs. The main difficulty in Lysozyme workflow lies in orchestrating external binaries and correctly enforcing dependencies between tasks, rather than in expressing task-level parallelism. Figure 1 analyzes the structure of the LLM-generated specifications that executed successfully. We distinguish between specifications that preserve multiple concurrent tasks (*parallel*) and those that reduce the workflow to a sequential execution (*serial*). The *raw* and *fixed* lines report the specifications generated by LLMs and the ones that were fixed. Two findings stand out. First, serial execution is particularly common in Nextflow. Only 11% of the raw speci-

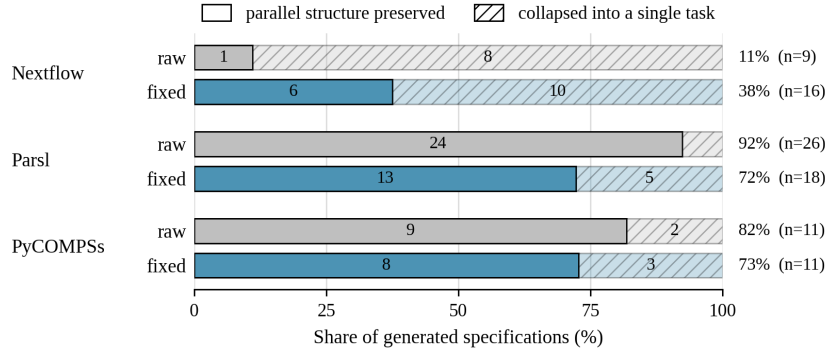


Figure 1. Preservation of parallel structure among the LLM-generated workflows.

fications (without minor corrections) preserve parallelism, compared with 92% for Parsl and 82% for PyCOMPSs. This result is consistent with the conversion failure mode discussed in Section 4.3. One special case is when the LLM transfers the task structure of a PyCOMPSs workflow to Nextflow without translating it into the processes and channel-based dataflow required by Nextflow. The resulting specification may execute, but as a single monolithic process rather than as a parallel workflow.

Second, repairing the specifications to make them executable does not necessarily guarantee parallelism. On Parsl, the proportion of parallel specifications decreases from 92% in the raw condition to 72% in the fixed condition; on PyCOMPSs, it decreases from 82% to 73%. The fixed workflow specifications include specifications that failed initially but became executable after repair, and these newly executable specifications are more likely to be serial. This behavior is expected under the restricted repair procedure ρ , which can correct configuration and syntax errors but cannot change task dependencies or workflow logic. Consequently, a specification that does not express concurrency cannot gain parallelism through repair. These results show that passing an execution test is not sufficient to establish that a generated workflow makes effective use of the workflow system and the distributed environment.

Figure 2 presents the ratio between the makespan of the LLM-generated workflow specification and that of the corresponding expert-developed execution of the workflows. The comparison is restricted to the workflow–system combinations for which at least one generated specification preserved the parallel structure and a matching reference execution was available on the same inputs. Even when the LLM generates a specification that supports parallelism, the median makespan exceeds that of the expert-developed specification in every combination evaluated, with individual cases reaching more than $5\times$ the reference. Only isolated executions matched or slightly outperformed the reference workflow. LLM-generated specifications can therefore identify opportunities for parallelism, but they do not exploit them as effectively as expert-developed specifications, *e.g.*, decorators are often added to code regions that are not compute-intensive, while regions that are remain unannotated.

Takeaway (Parallelism). Executability and parallelism are different properties, and ER and ER⁺ capture only the former. Evaluations that consider only whether an LLM-generated workflow executes can therefore overestimate its practical usefulness. This limitation is particularly important for dataflow-oriented systems such as Nextflow, where a workflow may execute successfully yet fail to guarantee (or preserve during translation) the parallel struc-

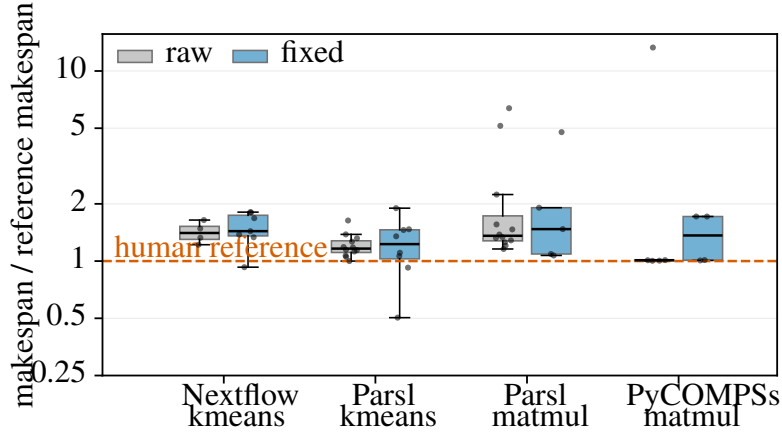


Figure 2. Makespan of LLM-generated workflows relative to the human reference.

ture. Even the specifications that do preserve it remain slower than the expert-developed.

5. Conclusions and Future Work

This paper presented a study of LLMs’ ability to specify and execute workflows in parallel workflow systems. An extensive evaluation with four LLM models and three parallel workflow systems, executed each generated workflow in a real environment rather than evaluating it solely by inspection. Overall, our results lead to three conclusions. First, lexical similarity and functional correctness are decoupled. Command-A-Plus achieved a BLEU score on Parsl comparable to GPT-5.5’s, yet failed to execute for every specification. Second, executability remains limited without human intervention. The highest average unrepaired execution rate across all configurations was only 46.7%, while the execution rate in the conversion task to Nextflow remained below 5%. A restricted repair procedure recovered a large fraction of these failures, suggesting that models generally capture the intended dataflow but struggle to reproduce system-specific configuration and API usage. Moreover, 10% of the executable K-Means and MatMul specifications computed a wrong result or nothing at all. Third, and most importantly for execution in HPC and cloud environments, successful execution does not imply parallelism. Approximately one-third of executable specifications scheduled all tasks sequentially, and repair was unable to recover parallelism. Moreover, even specifications that preserved concurrency were slower than the expert-developed reference workflows. As future work, we plan to extend our evaluation to additional workflow systems and larger distributed environments, investigate whether providing execution feedback enables iterative self-repair, and incorporate provenance data as an additional source of context for both generation and validation.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. The authors also thank CNPq and FAPERJ. This paper has been partially supported by the projects CEX2021-001148-S, and PID2023-147979NB-C21 from the MCIN/AEI and MICIU/AEI /10.13039/501100011033 and by FEDER, UE, by the Departament de Recerca i Universitats de la Generalitat de Catalunya, research group MPiEDist (2021 SGR 00412), and the Horizon Europe ODISSEE project (grant number 101188332). The views and opinions expressed are solely those of the author(s) and do not necessarily reflect those of the European Union. Neither the European Union nor the European Commission can be held responsible for them.

References

- Alam, K. and Roy, B. (2025). From prompt to pipeline: Large language models for scientific workflow development in bioinformatics. In *arXiv*, arXiv. arXiv.
- Amstutz, P., Crusoe, M. R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Lee, D., Ménager, H., Nedeljkovich, M., et al. (2016). Common workflow language, draft-3. *Figshare*.
- Babuji, Y., Woodard, A., Li, Z., Katz, D. S., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J. M., Foster, I., Wilde, M., and Chard, K. (2019). Parsl: Pervasive parallel programming in python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19)*, pages 25–36. ACM.
- Badia, R. M., Ayguade, E., and Labarta, J. (2017). Workflows for science: A challenge when facing the convergence of hpc and big data. *Supercomputing frontiers and innovations*, 4(1):27–47.
- Badia, R. M., Conejero, J., Ejarque, J., Lezzi, D., Lordan, F., and Sirvent, R. (2026). Programming task-based workflows with COMPSs. In *Workflow Systems for Large-Scale Scientific Data Analysis*. Berlin Universities Publishing.
- Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021). Evaluating large language models trained on code.
- Cohere (2026). Command a+. <https://cohere.com/command>. [Accessed 16-08-2026].
- Da Silva, R. F., Bard, D., Chard, K., De Witt, S., Foster, I. T., Gibbs, T., Goble, C., Godoy, W., Gustafsson, J., Haus, U.-U., et al. (2024). Workflows community summit 2024: Future trends and challenges in scientific workflows. *arXiv preprint arXiv:2410.14943*.
- de Oliveira, D., Ogasawara, E. S., Baião, F. A., and Mattoso, M. (2010). Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *IEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, 5-10 July, 2010*, pages 378–385. IEEE Computer Society.
- de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.
- Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., Ferreira da Silva, R., Livny, M., and Wenger, K. (2015). Pegasus, a workflow management system for science automation. *Future Gener. Comput. Syst.*, 46(C):17–35.
- Deepseek (2026). Deepseek v4. <https://deepseek.ai/deepseek-v4>. [Accessed 16-08-2026].
- Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., and Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319.

- Duque, A., Syed, A., Day, K. V., Berry, M. J., Katz, D. S., and Kindratenko, V. V. (2023). Leveraging large language models to build and execute computational workflows. *arXiv preprint arXiv:2312.07711*.
- Google (2026). Gemma 4. <https://deepmind.google/models/gemma/gemma-4/>. [Accessed 16-08-2026].
- Gu, Y., Cao, J., Guo, Y., Qian, S., and Guan, W. (2023). Plan, generate and match: Scientific workflow recommendation with large language models. In *Lecture Notes in Computer Science*, Lecture Notes in Computer Science, pages 86–102. Springer Nature Switzerland, Cham.
- Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015). A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493.
- Ljaljevic, S., Jorba, J., and Iserte, S. (2026). Exploring the role of large language models in high-performance computing programming: A survey. *Future Generation Computer Systems*, 184:108618.
- Miyashita, Y., Tung, P. K. M., and Barthélemy, J. (2024). Llm as hpc expert: Extending rag architecture for hpc data.
- OpenAI (2026). Gpt-5.5. <https://openai.com/index/introducing-gpt-5-5/>. [Accessed 16-08-2026].
- Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In Isabelle, P., Charniak, E., and Lin, D., editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.
- Popović, M. (2015). chrF: character n-gram F-score for automatic MT evaluation. In Bojar, O., Chatterjee, R., Federmann, C., Haddow, B., Hokamp, C., Huck, M., Logacheva, V., and Pecina, P., editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.
- Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.
- Sänger, M., De Mecquenem, N., Lewińska, K. E., Bountris, V., Lehmann, F., Leser, U., and Kosch, T. (2024). A qualitative assessment of using chatgpt as large language model for scientific workflow development. *GigaScience*, 13:giae030.
- Yildiz, O. and Peterka, T. (2025). Do large language models speak scientific workflows? In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC Workshops '25, page 2225–2233, New York, NY, USA. Association for Computing Machinery.
- Yin, J., Hines, J., Herron, E., Ghosal, T., Liu, H., Prentice, S., Lama, V., and Wang, F. (2025). chatHPC: Empowering HPC users with large language models. *J. Supercomput.*, 81(1).
- Zhao, W. X. et al. (2023). A survey of large language models. *arXiv:2303.18223*, 1(2).