Automated DevOps Pipeline Generation for GitHub Repositories using Large Language Models

Deep Mehta, Kartik Rawool, Subodh Gujar
North Carolina State University
Raleigh, North Carolina, USA
{dmmehta2,khrawool,sgujar}@ncsu.edu

ABSTRACT

Automating software development processes through the orchestration of GitHub Action workflows has revolutionized the efficiency and agility of software delivery pipelines. This paper presents a detailed investigation into the use of Large Language Models (LLMs) – specifically, GPT 3.5 and GPT 4 – to generate and evaluate GitHub Action workflows for DevOps tasks. Our methodology involves data collection from public GitHub repositories, prompt engineering for LLM utilization, and evaluation metrics encompassing exact match scores, BLEU scores, and a novel DevOps Aware score. The research scrutinizes the proficiency of GPT 3.5 and GPT 4 in generating GitHub workflows, while assessing the influence of various prompt elements in constructing the most efficient pipeline. Results indicate substantial advancements in GPT 4, particularly in DevOps awareness and syntax correctness. The research introduces a GitHub App built on Probot, empowering users to automate workflow generation within GitHub ecosystem. This study contributes insights into the evolving landscape of AI-driven automation in DevOps practices.

1 INTRODUCTION

Automation has become integral to modern software development, and GitHub Action workflows stand as a pivotal tool in orchestrating streamlined software delivery pipelines. This paper delves into an in-depth exploration of Large Language Models (LLMs), specifically GPT 3.5 and GPT 4, to generate and evaluate GitHub Action workflows tailored for DevOps tasks. Our research methodology involves the curation of pertinent data from public GitHub repositories and the engineering of prompts to guide LLMs in workflow generation. Evaluation metrics, including exact match scores, BLEU scores, and a newly introduced DevOps Aware score, assess the fidelity of generated workflows to DevOps best practices.

Central to this study is the comparative analysis between GPT 3.5 and GPT 4 in generating YAML files for DevOps tasks. Additionally, the research investigates the nuanced impacts of different prompt components on the performance of these models in workflow generation. A critical highlight of this study is the meticulous evaluation of the generated workflows, emphasizing not only syntactic correctness but also their alignment with DevOps methodologies.

Moreover, a pivotal contribution of this research is the development of a dedicated GitHub App. This App capitalizes on the capabilities of LLMs, allowing seamless workflow generation for any GitHub repository and immediate commit into the repository itself. This innovative tool empowers developers by automating the generation and integration of workflows directly into their projects, streamlining their software development lifecycle.

The outcomes of this research present promising advancements, particularly in the enhanced capabilities of GPT 4, exhibiting heightened DevOps awareness and improved syntactic accuracy. The GitHub App's development further signifies the practical application of AI-driven models in the realm of DevOps practices, albeit with identified limitations.

2 RELATED WORKS

The recent years the use of Large Language Models in code generation task have gained traction in the recent years. As outlined in the work [1], three predominant pre-training methods are identified in the domain of code language modeling. The initial category encompasses Left-to-Right Language Models, exemplified by models such as CodeGPT [2], CodeParrot [3], Codex [4] and CodeGen [5]. These models operate by predicting the probability of a given token based on preceding tokens, making them particularly beneficial for code completion tasks. The second category involves masked language models, including CodeBERT [6] and CuBERT [7], which leverage bidirectional information to generate whole-sequence representations. Notably, these models demonstrate notable performance in clone detection and defect detection applications. The third category revolves around encoder-decoder models, exemplified by models like CodeT5 [8] and PLBART [9]. These models employ an encoder to encode an input sequence and subsequently utilize a left-to-right language model for decoding an output sequence, proving effective in downstream tasks such as code commenting or natural language to code generation.

The study presented in the following paper [10] conducts experiments to assess the performance of ChatGPT across two distinct code generation tasks: text-to-code and code-to-code generation. The findings suggest that the overall generation performance can be significantly enhanced through meticulous prompt design. Various prompt types, such as task prompts, context prompts, processing prompts, and behavior prompts, were employed to facilitate the text-to-code generation task. The conclusion drawn emphasizes the pivotal role of prompt construction in optimizing the performance of ChatGPT in code generation scenarios.

The study [11] delves into the examination of the utilization and upkeep of GitHub Actions workflows within ten prominent GitHub repositories. Through manual inspection of 222 commits associated with workflow modifications, the researchers identified 11 distinct types of changes to these workflows. Additionally, they brought to light several shortcomings in the production and maintenance of GitHub Actions workflows, emphasizing the need for sufficient tooling to facilitate the creation, editing, refactoring, and debugging of GitHub Actions workflow files.

Previous research on GitHub Actions technology has indicated favorable reception among practitioners [12]. The adoption of GitHub Actions is evidenced by a rising count of monthly rejected pull requests and a simultaneous decline in the number of commits on merged pull requests. Common operations encompass continuous integration, diverse utilities (e.g., reading configuration files), deployment, publishing, and activities related to code quality and code review, among other types of actions.

In the work titled "Let's Supercharge the Workflows: An Empirical Study of GitHub Actions" [13], an in-depth analysis is conducted on the usage patterns of GitHub Actions. The study delves into the impact of GitHub Actions on various aspects, including commit frequency, and the efficiency of resolving pull requests and issues. Findings reveal that during the GitHub Actions configuration, developers are required to set multiple components, each presenting numerous alternatives. This complexity poses a challenge for less experienced developers in configuring a suitable GitHub Action Workflow. Furthermore, the study highlights the dependence of the GitHub Action configuration process on developers' personal experience, suggesting a need for assistive tools to facilitate the configuration of GitHub Actions.

This paper [14] addresses the gap of domain specific language models by focusing on Ansible-YAML, a widely employed markup language for IT Automation. The proposed solution, Ansible Wisdom, is a transformer-based model, uniquely trained on a dataset containing Ansible-YAML. To comprehensively assess its performance, the paper introduces two novel performance metrics tailored for YAML and Ansible, aimed at capturing the distinctive characteristics inherent to this specialized domain.

3 METHOD AND IMPLEMENTATION

In the section, we details the core design (i.e., prompt engineering) in Section 3.1 and implementation in Section 3.2 of our method DevOps-LLM-Bot.

3.1 Prompt Engineering

For generating a YAML file for a given repository, ideally, the more information about the repository in the prompt, the better performance is expected to be achieved. However, in practice, due to the token limit and also the imperfection of the LLM, we integrate our expert knowledge as the feature of the task into the prompt.

In this work, we customize the prompt with the following 3 parts as our first attempt. Particularly, for designing *context*, we consider 2 pieces of task-specific information, which are *file structure* and *default branch* based on our observation. For example, Figure 1 presents a real-world YAML file[1]. It carries both information about file structure (i.e. requirements.txt in the install step,) and branch (i.e. master while defining workflow triggers).

3.2 Tool

We develop a tool based on GitHub App[2]. Figure 2 describes the architecture of this app built on top of Probot framework, which responds to the following events:

[1]https://github.com/ajcr/rolling/blob/master/.github/workflows/python-package.yml
[2]https://github.com/apps/devops-llm-bot

```yaml
name: Python package

on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        python-version: ["3.7", "3.8", "3.9", "3.10", "3.11"]

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        python -m pip install pytest
        if [ -f requirements.txt ];
        then pip install -r requirements.txt; fi
    - name: Test with pytest
      run: |
        pytest
```

Figure 1: Example GitHub Actions Workflow

- **Issue Creation**: The app springs into action when a new issue is created with a title that commences with `@devops`. The context provided during this request includes the repository file structure and the default branch of the repository.
- **Pull Request Comment**: The app is also activated when a comment is made on a pull request, provided the comment starts with `@devops-llm-bot`. In this scenario, the GPT is invoked with the context of the existing workflow and all previous conversations that the user has had within that pull request.

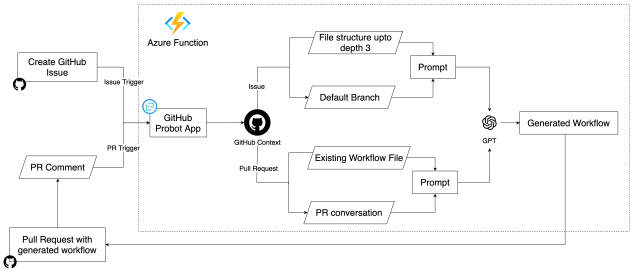

Figure 2: Architecture

The app runs on **Azure functions** and uses the **Octokit** client to interact with GitHub and get information about the repository, such as its structure and contents. In our work, **we develop an app that invoked the GPT 4 (*gpt-4-1106-preview*) chat completion API to generate workflow and raise a pull request for the developers to review, improvise by conversing with the bot and merge the workflow file**.

Component Name	Definition	Value
System Message	This outlines LLM's duties and expectations as a DevOps engineer. It emphasizes on the creation of workflows for building and testing applications, while excluding any steps for deployment across various platforms or environments.	You are a brilliant and meticulous DevOps engineer assigned to write a GitHub Actions workflow in YAML for the following Github Repository. When you write code, the code works on the first try, is syntactically perfect and is fully complete. The workflow should be able to build and run the application and run the tests if present in the repository. Do not include any deployment steps in the workflow.
User Message & Context	This provides insights about the GitHub repository, such as its file structure and the default branch. It also includes any specific requests from the user for generating the workflow.	Take into account the current repository's default branch, file structure and user requests: - Repository default branch - Repository tree - User requests

Table 1: Prompt Setting of DevOps-LLM-Bot

```
const completion = await openai.createChatCompletion({
        model: "gpt-4-1106-preview",
        messages: [
          {
            "role": "system",
            "content": systemMessage,
          },
          {
            "role": "user",
            "content": user_message,
          },
        ],
        functions: [{ name: "generate_workflow",
        parameters: output_schema}],
        function_call: {name: "generate_workflow"}
      });
```

The *output_schema* specifies the JSON format that the app is expecting the bot to respond to. Also, the schema helps to ensure that GPT only returns the workflow file contents in the workflow_file_content json field, which can be directly committed as a workflow.yml file in the repository.

```
    const output_schema = {
  "type": "object",
  "properties": {
    "workflow_file_content": {
      "type": "string"
    },
    "workflow_file_description": {
      "type": "string"
    },
    "commit_message": {
      "type": "string"
    }
  },
  "required": ["workflow_file_name",
  "workflow_file_content",
  "workflow_file_description", "commit_message"]
}
```

3.3 Usage

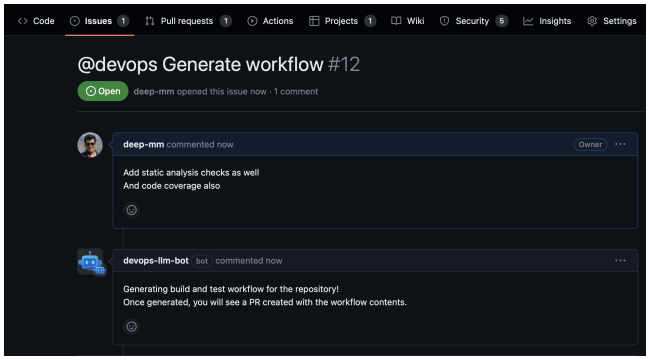

Figure 3: New GitHub Issue

As shown in Figure 3, the bot can automatically generate a workflow for a GitHub repository when a user creates a new issue with a title that begins with *@devops* and a body that specifies any custom requests. The bot replies to the issue with a message that indicates that it is working on the task. After the workflow is ready, the bot automatically commits it to a new branch and transforms the issue into a pull request that merges the new branch with the default branch of the repository.

Figure 4 demonstrates the direct interaction between the user and the LLM via the pull request, facilitating the necessary workflow updates. This interaction is designed to empower DevOps engineers by enabling them to generate the workflow without writing any code. It also allows them to verify the performance of the generated workflow, as it will automatically initiate upon the creation of the pull request. This ensures that the workflow operates as anticipated.

4 EVALUATION

4.1 Research Questions and Methodology

Our experiment setting is designed to answer the following research questions.

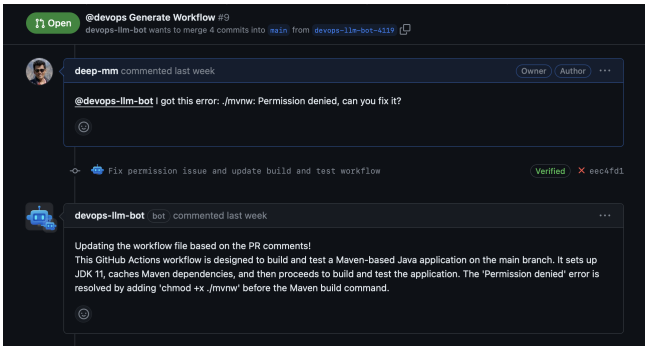

Figure 4: GitHub Pull Request Comment

- How the performance DevOps-LLM-Bot differ based on different models?
- How do the different components of the prompt contribute to the final performance of DevOps-LLM-Bot?
- How efficient is DevOps-LLM-Bot?

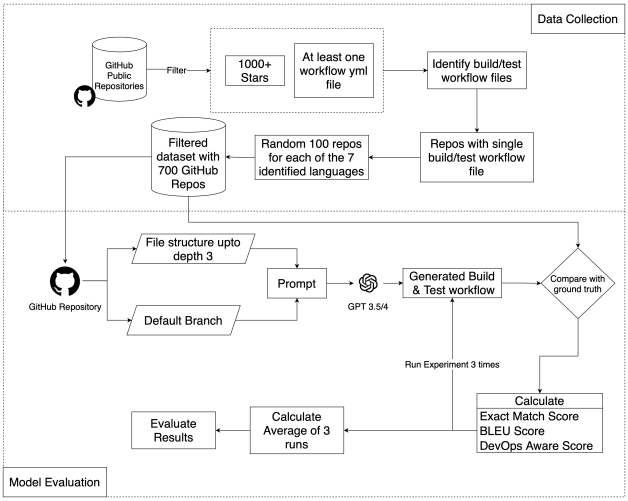

Figure 5: Framework for experimentation

Figure 5 presents the overview of our evaluation methodology. It demonstrates how we leverage public GitHub repositories to create a high-quality dataset for testing LLMs on DevOps tasks. In data collection (Section 4.2), we apply filter conditions to select relevant repositories. In model evaluation (Section 4.3), we generate build and test workflow with different settings in terms of prompt design and LLMs. We compare the generated workflows with the ground truth and evaluate them using multiple metrics in both automatic and manual manners.

Automatic Evaluation. Ideally, executing the generated workflow can be one of the accurate ways to measure their correctness. However, it would be time-consuming and not practical to evaluate each GitHub Actions Workflow by running it. Hence, we evaluate the pipeline based on the static similarity measurements between ground truth and the generated workflow. Specifically, we consider not only the exact match score, BLEU score, but also we introduce a customized metric DevOps Aware score (Section 4.3). To mitigate the randomness of the performance, we run each experiment three times for each setting and calculate the average scores.

Firstly, the syntax of the workflow is checked which checks if appropriate keys are used, the file follows workflow syntax and the structure is correct. Syntax validation of GitHub workflows is performed using actionlint [15] which is a static checker tool. For a given GitHub repository, there may be multiple workflow yaml files each for a different purposes such as build, run, test and deploy. The dataset collected only includes repositories which have a build and test workflow. For the evaluation, the workflow is checked for syntax if its valid or not and then the various metrics are calculated on the valid workflows.

Human Evaluation: In addition to automatic evaluation, we also conducted a manual evaluation to better understand and evaluate the quality of the generated workflow. Since we do not have a proven evaluation metric for yaml, a manual evaluation would help invalidate our novel DevOps Aware Score and also check how semantically our generated workflows are to ground truth. Hence, we sampled 324 generated workflows, with this sample size, we can maintain a confidence level of 95% and a 5% margin of error. The workflows were scored based on a 5-point Likert scale.

4.2 Data Preparation

To the best of our knowledge, our work is the first work that focuses on generating YAML file for a given code repository. Therefore, we construct the first dataset for the task with two stages, data collection and cleaning phase.

Data collection phase. We extract repository data from SEART-GHS tool (https://seart-ghs.si.usi.ch/) using specific criteria based on number of stars associated with each GitHub repository. The following filtering criteria is applied:

(1) **With at least 100 Stars**. To control the quality of our data, we apply the common indicator, the number of stars, to filter the low-quality repositories.
(2) **Contain at least 1 .yml or .yaml file within *.github/workflows* folder**. It indicates the presence of GitHub Actions.

To implement the above filtering criteria, we employ GitHub REST api to inspect repository contents and ascertain the existence of relevant files, ensuring the inclusion of repositories with GitHub Actions configurations.

Data cleaning phase. By design, GitHub Actions can be implemented for various purposes, such as build, test, deployment, etc[3]. Therefore, distinguishing the purpose of a specific GitHub Actions workflow file is challenging. To address this, we adopted an exhaustive list of build and test command lines tailored to specific programming languages. For example, in Python, we considered commands such as 'pip install' for package installation, 'pytest' for testing, 'python setup.py install' for package setup, 'tox' for testing multiple environments, 'pipenv install' for virtual environment setup, and 'django-admin test' for Django framework testing. The presence of any of these commands within a workflow file served as an indicator that the file was designed for build and test

[3]https://github.com/features/actions

purposes. However, even with this nuanced approach, our analysis revealed that only 16.56% of repositories had a single workflow file dedicated to building and testing, and 7.71% featured multiple such files, highlighting the complexity of accurately categorizing workflows based solely on contained commands.

To streamline our research, we focused on a wide range of programming languages, they are Java, Python, JavaScript, TypeScript, Kotlin, C#, & C++. We select 100 data points for each language to construct a manageable dataset. The final dataset includes essential information such as repository owner, repository name, and the ground truth workflow (i.e., the human-written YAML file).

4.3 Evaluation Metrics

In our work, we not only consider the widely-used metrics such as Exact Match (EM), BLEU [16], and Syntax Correct, but also introduce a new and specialized metric called *DevOps Aware Score* to evaluate the generated workflow. The reason is that those traditional metrics carry certain known limitations. EM completely ignores the semantically equivalent yet different pairs between the generated one and the ground truth. BLEU score is another popular metric but it heavily relies on n-grams and may not capture the overall meaning of the YAML file. Syntax Correctness is also an important metric but it only focuses on the syntax-wise.

DevOps Aware is a customized metric defined specifically to compare the semantic distance between two YAML files. The metric aims to leverage knowledge of GitHub Action Workflow syntax to assess the effectiveness of code sections that directly impact the output. It acknowledges that achieving the same objective can be accomplished through various predefined actions. The evaluation process involves examining the “jobs” section, which comprises “steps” specifying actions and commands for execution. We extract the build/test “jobs” content from both the ground truth and the generated workflow, parsing through the "steps" to compare the generated steps with their corresponding actual steps. The other steps from the ground truth are overlooked because these workflows may encompass more than just building and testing. At present, our assessment is primarily concerned with the Language Learning Model’s (LLM) ability to generate build and test actions. The comparison involves checking for keys such as “uses,” “runs,” or “name.” If both the generated step and the actual step contain "uses" or "runs," an Exact Match score is calculated. In other cases, the BLEU score is computed. The final score is the average of all the calculated scores.

Let’s take a look at the workflows depicted in Figure 6. In this scenario, we’re focusing on three specific steps: restore, build, and test. We compare these steps with the ground truth for evaluation. For the restore and build steps, both the ground truth and the generated workflows utilize the ‘run’ script. This results in an exact match score for these steps. However, the test step is a bit different. The ground truth uses a marketplace action, while the generated workflow uses a ‘run’ script. To evaluate this, we compute the BLEU score. Finally, to obtain the overall DevOps aware score, we average the three individual scores from the restore, build, and test steps. This gives us a comprehensive measure of how closely the generated workflow aligns with the ground truth.

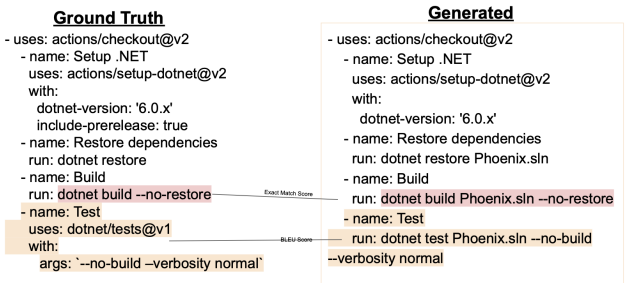

Figure 6: DevOps Aware Calculation

4.4 Result

Automatic Evaluation: In this comprehensive evaluation of GPT 3.5 and GPT 4 in generating GitHub Action workflows across multiple programming languages, several key metrics were employed to assess the models’ performance. Notably, GPT 4 exhibited a marked improvement over its predecessor, GPT 3.5, in key areas such as DevOps awareness, BLEU score, and, syntax correctness. The DevOps Aware Score, a critical metric reflecting the models’ understanding of DevOps practices in workflow generation, witnessed a substantial increase in GPT 4 across various languages, with notable improvements in C, C++, and Python. This suggests that GPT 4 has a more refined understanding of DevOps best practices, crucial for effective and seamless workflow automation.

Table 2: Comparison of GPT 3.5 and GPT 4

Models	DevOps Aware Score	BLEU Score	Exact Match	Syntax Correct (Valid / Total)
GPT 3.5	0.55	0.39	0.19	77.90 %
GPT 4	0.6	0.4	0.19	96.75 %

Table 3: Comparison of Valid syntax percentage

Languages	GPT 3.5	GPT 4
C#	87.33 %	99.33 %
C++	72.00 %	95.25 %
Java	66.33 %	97.6 %
Javascript	88.00 %	95.67 %
Kotlin	69.33 %	97.32 %
Python	72.33 %	95.00 %
Typescript	90.00 %	96.98 %

Furthermore, the syntax correctness results shed light on the models’ proficiency in generating syntactically accurate GitHub Action workflows. GPT 4 demonstrated a substantial leap in this aspect, particularly evident in the Syntax Correct percentage, where it achieved an impressive 96.75%, compared to GPT 3.5’s 77.90%. The detailed language-specific analysis in Table 2 3 corroborates this improvement, revealing significant strides in valid syntax percentages across languages. The discussion of Java’s lower syntax correctness

Table 4: Comparison of DevOps Aware Score

Languages	GPT 3.5	GPT 4
C#	0.44	0.53
C++	0.3	0.37
Java	0.55	0.6
Javascript	0.7	0.74
Kotlin	0.68	0.74
Python	0.45	0.49
Typescript	0.7	0.73

in GPT 3.5 underscores specific deficiencies, such as missing input definitions in the "actions/setup-java@v2" action. These findings collectively suggest that GPT 4 not only addresses previous syntactic limitations but also showcases enhanced language-specific understanding. In conclusion, the advancements seen in GPT 4 make a compelling case for its efficacy in generating GitHub Action workflows, offering a more reliable and syntactically accurate automation solution across diverse programming languages.

Manual Evaluation: We have checked for the correlation between the manually evaluated workflows based on the 5-point likert scale and the DevOps Aware Score. We choose Pearson correlation for its capacity to quantify linear associations, enabling nuanced analysis of DevOps and manual score relationships. Pearson correlation was computed to gauge linear associations, utilizing scipy.stats' pearsonr function in python script on DevOps and manual scores.

Table 5: Correlation between DevOps Aware Score & Manual Evaluation score

Languages	Correlation	Rows
C#	0.61	44
C++	0.24	42
Java	0.59	51
Javascript	0.48	53
Kotlin	0.54	52
Python	0.43	31
Typescript	0.54	51
Overall	0.61	324

Correlation Table 5 on DevOps aware and manual evaluation scores across programming languages reveals a robust positive association in C# (0.61), while C++ (0.24) suggests lower correlation, potentially linked to extensive workflow customization. The overall correlation of 0.61 across all languages provides profound insights into the intricate dynamics between automated DevOps assessments and manual evaluations which validates our novel evaluation metric for DevOps workflows.

5 HOW TO USE?

We've developed a GitHub App as part of our research, which you can access [here]. This app can be installed in any GitHub repository and used to generate GitHub build and test workflows for that repository.

You're welcome to explore the bot with this [example PR], or you can try it out in your own repository by following these steps:

(1) Install the GitHub app on your repository using the provided link.
(2) Create a new issue with a title beginning with @devops. The bot will use this to help generate your workflow. Any custom requests can be included in the issue description for the bot to consider while generating the workflow.
(3) You'll see a comment on your issue from the bot, indicating that your request is being processed.
(4) After about a minute, the issue will transform into a pull request and your workflow file will be generated.
(5) If you're satisfied with the workflow, you can merge the pull request.
(6) If you need modifications, add a comment in the pull request detailing the changes required. Make sure the comment begins with '@devops-llm-bot' to trigger the bot to consider your changes and re-generate the workflow.
(7) Once the workflow is generated, the bot will reply on the PR and add a new commit. Enjoy exploring the bot!

6 DISCUSSION

The conducted ablation study aimed to assess the impact of different contextual inputs on the performance of GPT-4 in generating GitHub Action Workflows. The three prompt settings employed were (1) Repository structure & dependency list, (2) Only Repository structure, and (3) Only Dependency list.

In Table 6, the comparison of DevOps Aware Scores across the different prompt settings revealed intriguing insights. Contrary to our initial hypothesis, the inclusion of dependencies did not significantly enhance the performance of GPT-4. The scores for scenarios with additional context (Repository structure + Dependency list) and those with only the Repository structure were comparable, with scores of 0.5 and 0.53, respectively. Interestingly, providing only the Dependency list resulted in a slightly lower score of 0.46. This suggests that, in our experimentation, the versioning information conveyed by project dependencies did not substantially contribute to the improvement of workflow generation.

Table 6: Comparison of DevOps Aware Score for ablation study

Languages	GPT 4
Repository structure + Dependency list	0.5
Only Repository structure	0.53
Only Dependency list	0.46

Further analysis, presented in Table 7, focused on the Valid Syntax percentage across the same prompt settings. Surprisingly, the results indicated a negligible difference in syntax validity among the three scenarios. Both Repository structure settings and the Dependency list setting demonstrated a high syntax validity percentage, with minimal variations at 96.90% and 96.88%, respectively. This implies that GPT-4 maintained consistent syntax generation capabilities across different contextual inputs.

Table 7: Comparison of Valid Syntax percentage for ablation study

Languages	GPT 4
Repository structure + Dependency list	96.90 %
Only Repository structure	96.90 %
Only Dependency list	96.88 %

In conclusion, the results suggest that, for the specific task of GitHub Action Workflow generation, the inclusion of additional context in the form of project dependencies did not yield a substantial improvement. On the contrary, the study underscores the importance of the repository structure as a critical factor influencing the generation of effective workflows. Therefore, when leveraging GPT-4 for GitHub Action Workflow generation, prioritizing information related to the repository structure in the input context is recommended for optimal performance.

7 CONCLUSION

In conclusion, the research outlined in this paper marks a significant milestone in the fusion of AI-driven capabilities, specifically GPT-4, within the realm of DevOps methodologies. The meticulous exploration and evaluation of GPT-4's capabilities showcased substantial advancements, particularly in its comprehension of DevOps practices and in generating syntactically accurate GitHub Action workflows.

The introduction of a dedicated GitHub App, empowered by Large Language Models, emerges as a significant contribution, allowing developers to automate workflow generation and seamlessly integrate it into their projects. This innovation streamlines the software development lifecycle, enabling developers to craft workflows efficiently and effectively.

Moreover, the comprehensive evaluation metrics utilized, encompassing exact match scores, BLEU scores, and the novel DevOps Aware score, offered deep insights into the fidelity of generated workflows to DevOps best practices. GPT-4 exhibited substantial improvements over GPT-3.5 across multiple languages, demonstrating a refined understanding of DevOps methodologies and providing a reliable solution for accurate workflow automation.

Looking ahead, the future holds promising avenues for further enhancements. Fine-tuning models, experimenting with different models and prompt combinations, and extending the scope to include deployment steps within GitHub workflows are potential areas for future work. These endeavors could fortify the efficacy and versatility of AI-driven models in enhancing DevOps practices, ushering in a new era of efficiency and effectiveness in software development pipelines.

In essence, this paper's contributions serve as a testament to the evolving landscape of AI-driven automation in DevOps, laying a strong foundation for future advancements in harnessing language models to augment DevOps efficiency and effectiveness.

REFERENCES

[1] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10, 2022.

[2] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

[3] Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language processing with transformers.* " O'Reilly Media, Inc.", 2022.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

[6] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[7] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. Learning and evaluating contextual embedding of source code. In *International conference on machine learning*, pages 5110–5121. PMLR, 2020.

[8] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

[9] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333*, 2021.

[10] Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. Improving chatgpt prompt for code generation. *arXiv preprint arXiv:2305.08360*, 2023.

[11] Pablo Valenzuela-Toledo and Alexandre Bergel. Evolution of github action workflows. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 123–127. IEEE, 2022.

[12] Timothy Kinsman, Mairieli Wessel, Marco A Gerosa, and Christoph Treude. How do software developers use github actions to automate their workflows? In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 420–431. IEEE, 2021.

[13] Tingting Chen, Yang Zhang, Shu Chen, Tao Wang, and Yiwen Wu. Let's supercharge the workflows: An empirical study of github actions. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 01–10. IEEE, 2021.

[14] Saurabh Pujar, Luca Buratti, Xiaojie Guo, Nicolas Dupuis, Burn Lewis, Sahil Suneja, Atin Sood, Ganesh Nalawade, Matt Jones, Alessandro Morari, et al. Automated code generation for information technology tasks in yaml through large language models. *arXiv preprint arXiv:2305.02783*, 2023.

[15] Linda_pp (rhysd). Actionlint - static checker for github actions workflow files. retrieved from https://github.com/rhysd/actionlintactionlint.

[16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.