Analysis of Language Models for Pyomo Code Generation: Benchmarking and Fine-Tuning

Evaluating StarCoder and CodeLlama

The final result on LangChain and GPT 3.5 Turbo

Prepared for
Data Science Team
Edgecom.ai

Prepared by
Azin Katiraee

July 16, 2024

Contact Information
azinkatiraie@gmail.com
LinkedIn

Executive Summary

This report outlines the results of the LLM Engineer - Code Challenge, aimed at enhancing the application of language models to automatically update and manage Pyomo code through a conversational interface. Initially, efforts were made to refine the performance of advanced models such as StarCoder and CodeLlama; however, due to limited GPU resources, these attempts did not yield full results. Consequently, the project shifted focus to the LangChain and GPT 3.5 Turbo model for an efficient solution and reaching to 84% Average BLEU Score.

Introduction

In the rapidly evolving field of optimization and automation, this report documents the integration and evaluation of language models through the LLM Engineer - Code Challenge, specifically focusing on the implementation of LangChain combined with GPT-3.5 Turbo. This innovative approach was tested on a synthetic dataset to determine its effectiveness. It is important to note that this implementation does not support data pre-processing, training, post-processing, or continuous learning. However, the use of OpenAI APIs offers a powerful and cost-effective alternative, capable of handling diverse scenarios without the extensive demands of training and continuous data updates required by open-source models. Given the constraints of time and available resources, the integration of LangChain with GPT-3.5 Turbo represents the most efficient solution for this project, striking a balance between performance and resource utilization.

Objective

The primary objective of this project is to evaluate the effectiveness of integrating LangChain with GPT-3.5 Turbo in enhancing the efficiency of code generation within Pyomo models. The specific goals are to measure performance improvements, assess user satisfaction with the interactive model, and identify challenges faced during the implementation process.

Methodology

Data Collection Methods: Synthetic data was generated using ChatGPT, which created various prompts and the corresponding Pyomo code stored in JSON files.
Tools and Techniques Used: The system's core utilizes the langchain_openai library along with the langchain.prompts module interfaced with the GPT-3.5 Turbo model from OpenAI. A predefined Pyomo model template served as the static foundation for all interactions, ensuring a consistent base for all modifications prompted by user input.
Procedures Followed: The operational flow involved initializing the LangChain environment with the necessary configurations, including the OpenAI API key setup. The interact_with_chain function was pivotal, processing user inputs to dynamically alter the Pyomo model template by inserting new constraints as specified. This was accomplished through a tailored prompt template that guided the language model to accurately interpret and execute the modifications without altering the logical structure of the Pyomo model.

Justification for the Chosen Methods: The integration of LangChain with GPT-3.5 Turbo was primarily driven by the need for an efficient, resource-conscious solution capable of rapid and accurate code modification. This setup minimized the need for extensive computational resources and training, typically required by more conventional machine learning approaches. The chosen methodology capitalized on the advanced capabilities of AI to provide a flexible, robust platform for optimizing and customizing Pyomo models, thereby enhancing the automation and efficiency of optimization processes.

Model Selection

For the initial phase of this project, StarCoder and CodeLlama were selected based on their demonstrated proficiency in handling code-related tasks, as highlighted in the article "Exploring Large Language Models for Code Explanation." These models are especially effective in zero-shot learning scenarios, where there are no pre-labeled examples, making them ideal for dynamic and unpredictable coding environments. Their fine-tuned capabilities enable a deep understanding of programming languages, which is essential for tasks like code summarization and explanation.

However, the project's requirements evolved to prioritize not only accuracy but also efficiency and scalability in real-time applications. This led to the selection of LangChain coupled with GPT-3.5 Turbo for the second phase of the project. The decision to use LangChain and GPT-3.5 Turbo was influenced by several key factors:

Efficiency and Speed: GPT-3.5 Turbo is optimized for quicker response times compared to earlier models. This speed is crucial for the interactive nature of the project, where user inputs must be processed and reflected in the Pyomo model almost instantaneously.

Advanced Understanding and Contextual Awareness: GPT-3.5 Turbo, being one of the latest iterations of OpenAI's models, offers enhanced understanding and better contextual awareness. This is vital for accurately interpreting complex user instructions and converting them into correct Pyomo code.

Resource Optimization: Unlike StarCoder and CodeLlama, which require substantial computational resources for optimal performance, GPT-3.5 Turbo operates efficiently under constrained resource settings. This makes it a more viable option for scenarios with limited GPU availability.

I**ntegration Flexibility**: LangChain provides a flexible framework that seamlessly integrates with GPT-3.5 Turbo, facilitating the dynamic insertion of user-defined constraints into the Pyomo model. This integration supports the project's goal of enabling non-experts to modify optimization models without deep technical knowledge of the underlying code.

Scalability: The combination of LangChain and GPT-3.5 Turbo supports scalability, allowing the system to handle an increasing number of user queries and more complex modifications without a loss in performance.

Fine-Tuning Techniques

For the initial phase of this project, two advanced fine-tuning techniques are utilized: **Parameter-Efficient Fine-Tuning (PEFT) and prefix fine-tuning**. These techniques are designed to enhance model performance while efficiently managing computational resources.

Parameter-Efficient Fine-Tuning (PEFT) offers a strategic approach to model training, where only a small fraction of the model's parameters are updated. This method is particularly useful in scenarios where computational resources are limited, or when training data is not abundant. PEFT leverages the pre-trained nature of large language models, allowing for rapid adaptation to specific tasks without the need for extensive retraining of the entire model network. This technique not only speeds up the fine-tuning process but also reduces the risk of overfitting, making it ideal for refining models on the specialized task of generating code.

Prefix fine-tuning, on the other hand, involves prepending a small set of trainable parameters (prefixes) to the inputs of each layer in the transformer model. These prefixes act as task-specific adjustments, guiding the model's attention mechanism to focus on features that are most relevant for the code explanation task. By altering how the model processes input without changing the underlying model architecture, prefix fine-tuning offers a flexible and efficient way to tailor the model's output towards desired outcomes, ensuring that the generated explanations are both accurate and contextually appropriate.

Utilizing these fine-tuning techniques, I aim to effectively adapt the underlying capabilities of StarCoder and CodeLlama to meet the specific requirements of this challenge.

The Mixed precision training was used to optimize the performance and efficiency of my model. Mixed precision training involves using both 16-bit (half precision) and 32-bit (single precision) floating-point types during computations. The fine-tuning did not have a result due to lack GPU as mentioned.

Evaluation Metrics

The evaluation of the LangChain and GPT-3.5 Turbo integration within this project was conducted using the BLEU score, a common metric in natural language processing used to measure the accuracy of generated text against a reference text. This metric was chosen due to its relevance in assessing the linguistic and syntactical similarity between the generated Pyomo code and the expected correct code provided in a synthetic dataset.

Experiments and Results

In the evaluation phase, the system was tasked with processing user inputs to modify a predefined Pyomo model by adding specified constraints. For instance, in one test scenario, the system successfully integrated the constraint "2X + 6Y > 30" into the existing model, demonstrating the model's capability to interpret and apply user-provided mathematical expressions accurately. The system's performance was quantitatively assessed using 100 synthetic samples, where the BLEU score—a measure of textual similarity between the generated and expected code—averaged 84%. This high score indicates that the generated code closely matched the reference code, confirming the

effectiveness of the LangChain and GPT-3.5 Turbo integration in producing syntactically and semantically correct modifications to the Pyomo model.

```
Please enter the details you want to add to the code: add a constraint of 2X + 6Y > 30 in our model
from pyomo.environ import *

model = ConcreteModel()
model.x = Var(domain=NonNegativeReals)
model.y = Var(domain=NonNegativeReals)
model.obj = Objective(expr=2 * model.x + 3 * model.y, sense=maximize)

# Define constraints
model.constraint1 = Constraint(expr=model.x + 2 * model.y <= 8)
model.constraint2 = Constraint(expr=3 * model.x + 2 * model.y <= 12)
model.constraint3 = Constraint(expr=2 * model.x + 6 * model.y > 30)

# Solve the model
solver = SolverFactory('glpk')
result = solver.solve(model)
```

```
(venv) (base) azinkatiraee@Azins-MacBook-Pro-2 pyomo_LLM_challenge % python evaluate.py

Average BLEU Score: 0.84
```

Conclusion and Recommendations

The project successfully demonstrated that integrating LangChain with GPT-3.5 Turbo can effectively automate the process of modifying optimization models in response to user inputs. The average BLEU score of 84% across 100 samples illustrates that the system can reliably generate accurate and relevant modifications to the Pyomo code. However, there is room for further improvement, particularly in enhancing the system's ability to handle more complex or ambiguous user inputs.

Recommendations for future work include:

1. **Enhancing Model Training:** While the current setup does not involve model training, incorporating a training phase could improve the model's accuracy and robustness, especially for more nuanced code modifications.
2. **Expanding Test Cases**: Increasing the diversity and complexity of test scenarios could help in better understanding the limitations of the current system and refining its capabilities.
3. **Optimizing Computational Efficiency**: Further optimization of the computational resources could reduce response times and increase the system's scalability.
4. **Continuous Learning Implementation**: Introducing mechanisms for continuous learning could enable the system to adapt to new user behaviors and evolve optimization tasks over time.

These steps would help in maximizing the potential of AI-driven automation in optimization modeling.

References

Bhattacharya, P., Chakraborty, M., Palepu, K. N. S. N., Pandey, V., Dindorkar, I., Rajpurohit, R., & Gupta, R. (2023). *Exploring Large Language Models for Code Explanation*. https://arxiv.org/abs/2310.16673

Lozhkov, A., Li, R., Ben Allal, L., Cassano, F., Lamy-Poirier, J., Tazi, N., Tang, A., ... de Vries, H. (2024). StarCoder 2 and The Stack v2: *The Next Generation*. https://huggingface.co/bigcode/starcoder2-7b

CodeLlama. (n.d.). CodeLlama-7B-HF. https://huggingface.co/codellama/CodeLlama-7b-hf