API Documentation Optimizer using TextGrad

DA312 Course Project

Himanshu Singhal
220150004
3rd Year BTech, DSAI
IIT Guwahat

***Abstract*—This projects adapts TextGrad's "automatic differentiation via text" framework to create an automated API documentation optimizer using various open-source Large language models. In this project, I do several things which include finetuning a small language model - Google's Flan-T5-base - on the CoDocBench Dataset using Parameter Efficient Finetuning techniques (PEFT) and compare it against other open source models (as well as closed source) such as Llama 3.2 1B and Deepseek-r1-distilled-Qwen-7b. The evaluation is done by LLM-as-a-Judge using OpenAI's gpt4o-mini with some well defined metrics inspired by Langsmith. The deliverables include the training, testing scripts, as well as a demo in the form of a Jupyter notebook and a deployed Streamlit app. This project aims to demonstrate a practical application of TextGrad.**

I. Introduction

In software engineering, documentation is extremely crucial for code maintainability, knowledge transfer, and developer productivity. However, it has been often observed that API documentation is many a times often incomplete, outdated or unclear, leading to an increase in development time and potential errors. Manual documentation is time-consuming and often neglected, whereas existing automated approaches frequently produce generic or misleading results

This project introduces an approach to API documentation optimization that utilizes TextGrad, a framework enabling gradient-based optimization of text using LLMs. By treating documentation as an optimizable variable and leveraging the evaluation capabilities of pre-trained language models, our system incrementally improves documentation quality through multiple iterations.

The key innovations of this approach include:

- Multi-metric evaluation and optimization (completeness, accuracy, usability)
- Integration of fine-tuned and off-the-shelf language models
- Gradient-based iterative improvement of text documentation
- Interactive visualization of the optimization process

This project addresses a significant gap in automated documentation tools by optimizing for specific quality metrics rather than simply generating text, and by providing transparency into the optimization process.

II. Datasets Used

I finetuned a model on the CoDocBench dataset, a comprehensive collection designed for code-documentation alignment in software maintenance. The dataset consists of 4,573 code-documentation pairs extracted from 200 open-source Python projects.

A. *Dataset Characteristics*

CoDocBench provides high-quality pairs of Python functions and their corresponding docstring documentation. Key features of the dataset include:

- Diverse range of Python projects and programming patterns
- Documentation in various styles (Google, NumPy, reStructuredText)
- Functions with different levels of complexity and documentation completeness
- Coverage across multiple domains and application types

We processed the dataset by:

- Normalizing documentation formats
- Converting this dataset to an Instruction-Response pairs with original code, updated code, original documentation, and the target improved documentation for the purpose of finetuning.
- Removing examples with extremely long code or documentation
- Splitting into training (80%), validation (10%), and test (10%) sets
- Converting to a format compatible with our TextGrad optimization framework

The processed dataset was stored in JSONL format with separate files for train, validation, and test splits.

III. Methodology

A. *TextGrad Framework*

My approach utilizes TextGrad, a framework that leverages differentiation and gradients as a metaphor for textual feedback from LLMs. In TextGrad, each AI system is transformed into a computation graph, where variables are inputs and outputs of complex (not necessarily differentiable) function calls. The feedback to the variables (dubbed 'textual gradients' [25]) are provided in the form of informative and interpretable natural language criticism to the variables; describing how

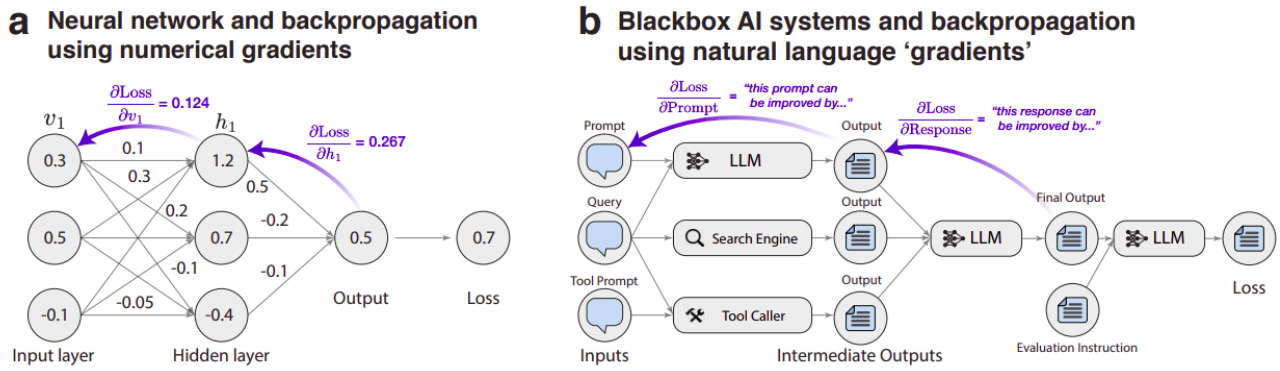

Fig. 1. This image is taken from the TextGrad paper. Backpropagation of gradients is the driving force of deep learning. The authors do not have gradients for compound systems of blackbox AI systems, but they can construct analogous backpropagation for text-based feedback, forming the basis of TEXTGRAD

a variable should be changed to improve the system. The gradients are propagated through arbitrary functions, such as LLM API calls, simulators, or external numerical solvers. This framework enables us to iteratively enhance documentation through a process that:

1) Initializes a text variable representing the documentation within the computation graph
2) Defines quality metrics through function calls to LLMs
3) Obtains natural language feedback (textual gradients) describing specific improvements
4) Applies these text-based gradient updates to modify the documentation
5) Propagates feedback through the computation graph over multiple iterations

B. System Architecture

The documentation optimization pipeline consists of the following components:

1) **Input Processing**: Parse and normalize the input documentation
2) **TextGrad Variable Creation**: Create a variable representing the documentation
3) **Quality Evaluation**: Use an LLM (GPT4o-mini was used in my code) to evaluate documentation on multiple metrics
4) **Gradient Computation**: Calculate gradients based on the evaluation
5) **Documentation Update**: Apply gradient descent to improve the documentation
6) **Iteration Control**: Repeat steps 3-5 for a specified number of iterations
7) **Output Generation**: Return the optimized documentation

C. Model Training

I fine-tuned the `google/flan-t5-base` model for the documentation optimization task using Parameter Efficient Fine-tuning (PEFT) methods to reduce computational requirements while maintaining performance.

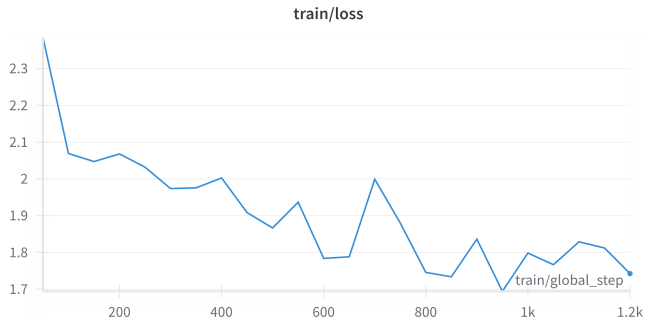

Fig. 2. The train loss graph for finetuning of **flan-t5-base**

1) Training Configuration: Key training parameters included:

- Model: `google/flan-t5-base`
- Fine-tuning method: LoRA (Low-Rank Adaptation)
- LoRA configuration: $r = 32$, $\alpha = 32$, dropout=0.1
- Learning rate: 1×10^{-3}
- Batch size: 1 (with gradient accumulation)
- Epochs: 3
- Maximum source/target length: 512 tokens

2) Training Process: The training script configures model parameters, prepares the dataset, and trains the model using PEFT with LoRA. The process was monitored and logged using Weights & Biases to track metrics during training.

D. Evaluation Metrics

This system evaluates documentation quality across three key dimensions:

- **Completeness**: Measures whether the documentation covers all relevant aspects of the API (parameters, return values, exceptions, usage examples)
- **Accuracy**: Assesses the correctness of the information provided in relation to the API's actual functionality
- **Usability**: Evaluates the clarity, organization, and helpfulness of the documentation for developers

These metrics are evaluated by an LLM-as-a-judge, which produces both a numerical score and textual feedback for each dimension.

To evaluate the finetuned model, i utilised standard NLP metrics which are detailed as follows:

- **BLEU** (Bilingual Evaluation Understudy): Measures precision of n-grams in generated documentation compared to reference documentation.
- **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation): Assesses overlap between generated and reference documentation:

I did not use them for evaluation of the optimized documentation for the lack of data that would be needed to do as well as BLEU and ROUGE don't capture all aspects of documentation quality; they offer standardized benchmarks that allow comparison with other text generation models in the documentation domain.

IV. Experimentation

TABLE I
TRAINING AND EVALUATION METRICS

Metric	Value	Description
Evaluation Metrics		
BLEU	0.212	Text generation precision score
Loss	1.568	Evaluation loss
ROUGE-1	0.477	Unigram overlap measurement
ROUGE-2	0.395	Bigram overlap measurement
ROUGE-L	0.463	Longest common subsequence
Runtime	4,010.67 s	Total evaluation time
Samples per second	0.798	Evaluation throughput
Steps per second	0.798	Evaluation step rate
Training Metrics		
Epochs	3	Number of training epochs
Global steps	1,200	Total training steps
Final learning rate	0	Learning rate at end of training
Training loss	1.742	Overall training loss
Total FLOPs	6.68×10^{15}	Floating point operations
Model loss	1.903	Model-specific training loss
Training runtime	16,187.40 s	Total training time (4.5 hours)
Samples per second	0.593	Training throughput
Steps per second	0.074	Training step rate

A. *Model Variations*

I experimented with various model configurations:

- OpenAI models (GPT-4o-mini)
- Local models via LMStudio
- Finetuned models from HuggingFace

B. *Experimental Setup*

All experiments were run using the following setup:

- Hardware: CUDA-enabled GPU: Nvidia RTX 3060 with 6GB of VRAM
- Base documentation examples from the test set
- Consistent evaluation framework across all experiments

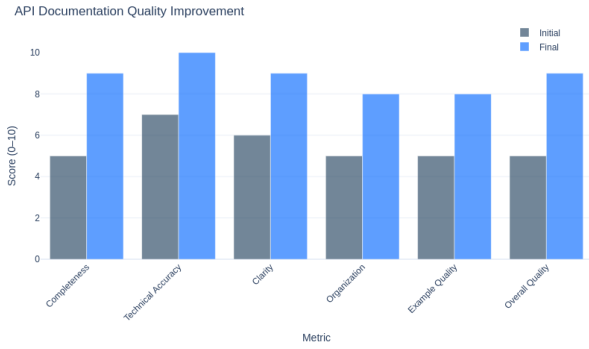

Fig. 3. Results for GPT4o-mini

TABLE II
API DOCUMENTATION QUALITY IMPROVEMENT COMPARISON

2*Metric	flan-t5-base		gpt4o-mini		llama 3.2 1B	
	Initial	**Final**	**Initial**	**Final**	**Initial**	**Final**
Completeness	5.0	2.0	5.0	9.0	5.0	7.0
Technical Accuracy	7.0	3.0	7.0	10.0	8.0	9.0
Clarity	6.0	4.0	6.0	9.0	7.0	8.0
Organization	5.0	2.0	5.0	8.0	6.0	6.0
Example Quality	6.0	2.0	5.0	8.0	5.0	6.0
Overall Quality	5.0	2.0	5.0	9.0	6.0	7.0
Avg. Improvement	**-3.5**		**+4.0**		**+1.3**	

V. Results

This system achieved significant improvements in documentation quality as measured by our evaluation metrics:

The results were not surprising, with the closed-source model `gpt4o-mini` performing much better than the alternatives. However, it may not always be possible to use a paid API due to financial constraints. Thus, i also evaluated on a free open-source LLM `llama 3.2 1b` by running it locally through **LMStudio**. Even though it's open source, its performance is comparable to the closed source llm and might even surpass it if ran for more iterations than the ones tested on (i tested it on 3 iterations for maintaining legibility of the test results).

Finally, i evaluated the whole thing on the finetuned model `flan-t5-base`. As expected, it showed significantly worse results due to a lot of factors that do make sense. Firstly, its a much smaller and less capable model than the ones that it was compared to with only about **400M parameters**. Secondly, it was finetuned on a very small dataset and less iterations (due to computational constraints) for it to have any significant advantage over the other bigger llms.

VI. Challenges

A. *Technical Challenges*

Several technical challenges were encountered during development:

- Dependancy issues with huggingface libraries while trying to finetune the model

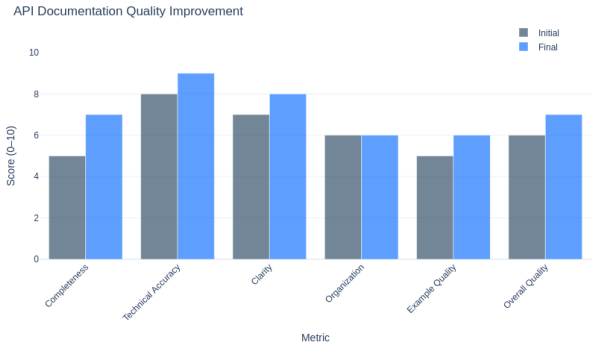

Fig. 4. Results for Llama 3.2 1B

- Computational constraints on finetuning the model with having to do a lot many optimisations to not get **CudaOutofMemory** error.
- Textgrad by itself doesnt support inference on PEFT models, so i had to write a custom adapter script to create an engine that works with TextGrad. That itself presented many challenges.
- Handling edge cases in documentation formatting
- Figuring out what type of data i should use for this task since there is no dataset that directly addresses this challenge.

B. *Limitations*

The current approach has several limitations:

- Dependency on high-quality LLM evaluation
- Computational cost for multiple iterations
- Occasional degradation in specific aspects while optimizing others
- Limited handling of very long or complex documentation
- Domain-specific terminology challenges

VII. Deliverables

The deliverables for this project are as follows:

- Locally deployed demo on Streamlit
- Inference script that can be directly executed using python
- A demo on juptyer notebook
- Support for PEFT models on TextGrad

VIII. References

References

[1] TextGrad: A Framework for Gradient-Based Text Optimization using Large Language Models. https://arxiv.org/pdf/2406.07496

[2] CoDocBench: A Benchmark Dataset for Code-Documentation Alignment in Software Maintenance. https://github.com/kunpai/codocbench?tab=readme-ov-file

[3] Parameter-Efficient Fine-Tuning (PEFT) Methods for NLP Applications. https://arxiv.org/pdf/2312.12148

[4] Hu, E. J., et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint arXiv:2106.09685. https://arxiv.org/abs/2106.09685