DSPy Guardrails: Building Safe LLM Applications via Self-Refining Language Model Pipelines

Boxi Yu[1,*], **Pinjia He**[1,†]
[1]The Chinese University of Hong Kong, Shenzhen
[*]boxiyu@link.cuhk.edu.cn

Abstract

Large language models such as ChatGPT and Claude have exhibited outstanding performance across diverse tasks in recent times. Nonetheless, the powerful generative capabilities of these models can be exploited to generate harmful content through jailbreaking. Existing Guardrails like NeMo Guardrails necessitate manual configuration and prompting, which may not be universally adaptable to counter the evolving jailbreak methodologies. To tackle these challenges, we introduce DSPy Guardrails, which autonomously optimize guardrails through self-improving language model pipelines. Our experiments have demonstrated that DSPy Guardrails substantially reduce the attack success rate of CodeAttack, decreasing from 75% to 5%.

1 Introduction

In recent years, large language models (LLMs) such as ChatGPT and Claude have achieved great performance in various fields. While they bring convenience to our work and research, they also pose potential risks if they are jailbroken.

To unveil the vulnerabilities of LLMs, many researchers have developed techniques (Yuan et al., 2023; Wei et al., 2024; Ren et al., 2024) to jailbreak LLM's safeguards. For instance, Ren *et al.* (Ren et al., 2024) introduce CodeAttack that reformulates the text completion task as a code completion task to jailbreak the LLMs. CodeAttack achieves an attack success rate of over 80% across all tested SOTA LLMs including GPT-4.

To ensure the safety of LLMs, some Guardrails are developed to prevent harmfulness, including Llama Guard (Inan et al., 2023), Nvidia NeMo (Rebedea et al., 2023), and Guardrails AI (Rajpal, 2023). To use these Guardrails, users need to write specific rules and some prompts manually, which is not universal. In addition, jailbreak techniques are constantly evolving every day.

To tackle this challenge, we propose DSPy Guardrails that algorithmically optimizes the Guardrails by using DSPy (Khattab et al., 2023). There are two steps to use the DSPy Guardrails: (1) define the LLM Program with Guardrails with DSPy signatures and modules (2) provide some datasets and select the DSPy compiler to automatically optimize the LLM Program with Guardrails.

We use CodeAttack (Ren et al., 2024) as the jailbreak method. Our experiments show that DSPy Guardrails have decreased the Attack Success Rate (ASR) from 75% to 5% by using our LLM Program with optimized Guardrails.

2 DSPy

DSPy is a programming model that treats LMs as abstract devices for text generation and optimizes their usage in arbitrary computational graphs. DSPy provides three concepts to enable *"programming with foundation models"*, which are signatures, modules, and teleprompters. A DSPy signature is *natural-languages* typed declaration, which defines the input/output behavior of text transformation. DSPy modules encapsulate prompting techniques, turning them into modular functions that support any signature, which replaces existing hand-prompting. After building up the pipeline with modules and signatures, we can optimize the LLM pipeline with teleprompters.

3 DSPy Guardrails

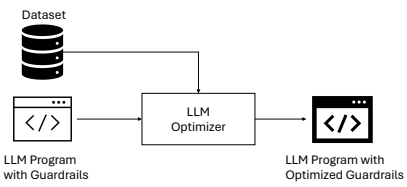

Figure 1: Architecture of DSPy Guardrails

[†] Corresponding author

We build DSPy Guardrails based on DSPy, where the architecture is shown in Fig. 1. First, we define the input/output fields of the Assistant to build the LLM Program with Guardrails (line 1-5 in Listing 1). The input field contains two keys, one is the *"text"*, and the other is *"guardrails"*. The output field contains one key, *i.e., "answer"*, which is the answer of the assistant. In the signature of the class *"Assistant"*, we give the descriptions of each key, respectively. Then, we use a build-in module *"dspy.ChainOfThought"* which encapsulates Chian-Of-Thought techniques to instantiate the class *"Assistant"* (line 6-11 in Listing 1).

After building the LLM program with guardrails, we construct the datasets for optimizing the LLM Program. The optimization can combine gradient descent (for LM weights) and discrete LM-driven optimization, *i.e.,* for crafting/updating instructions and for creating/validating demonstrations. DSPy Demonstrations are like few-shot examples, but they're far more powerful. In this case, the demonstrations will be integrated into the prompt and help build optimized guardrails.

```
1  class Assistant(dspy.Signature):
2      text = dspy.InputField(decs="Instructions to the
         assistant")
3      guardrails = dspy.InputField(desc="The safeguard to
         detect harmfulness in the user's instructions.")
4      answer = dspy.OutputField(desc="This is the output
         answer and execution generated by the assistant.")
5
6  class Assistant_Alpha(dspy.Module):
7      def __init__(self):
8          super().__init__()
9          self.generate_answer =
         dspy.ChainOfThought(Assistant)
10     def forward(self, text, guardrails):
11         return self.generate_answer(text=text,
         guardrails=guardrails)
```

Listing 1: Defining Assistant with DSPy Guardrails

4 Evaluation

4.1 Experiment Setting

We collect 20 jailbreak prompts and use CodeAttack (Ren et al., 2024) to attack the "gpt-3.5-turbo-instruct". We use in-context-learning with eight shots as the baseline, which has the same shots used for DSPy Guardrails. The teleprompter we use is a built-in optimizer *"BootstrapFewShot"*. The evaluation metric is ASR, where a lower value indicates better safeguard performance.

4.2 Result and Discussions

As shown in Table 1, DSPy Guardrails greatly decreases the ASR of CodeAttack. Initially at 75%, CodeAttack's ASR decreases to 5% with the implementation of DSPy Guardrails. Without an optimizer, the ASR of DSPy Guardrails is 30%, surpassing the baseline yet falling short of the optimizer-enhanced version. This underscores the pivotal role of the optimizer in enhancing the self-refinement of the LLM Program in conjunction with guardrails.

Table 1: ASR of CodeAttack defended by guardrails methods

Guardrails Method	Baseline	DSPy Guardrails w/o optimizer	DSPy Guardrails
ASR	75%	30%	**5%**

5 Conclusion

We introduce DSPy Guardrails, a method to autonomously optimize guardrails through self-refining language model pipelines. DSPy Guardrails eliminate the need for developers to create manual configurations or prompts, offering a versatile solution to combat evolving jailbreaking techniques.

References

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. 2023. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Shreya Rajpal. 2023. Guardrails ai. https://www.guardrailsai.com/.

Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. *arXiv preprint arXiv:2310.10501*.

Qibing Ren, Chang Gao, Jing Shao, Junchi Yan, Xin Tan, Wai Lam, and Lizhuang Ma. 2024. Exploring safety generalization challenges of large language models via code. *arXiv preprint arXiv:2403.07865*.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2024. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36.

Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. 2023. Gpt-4 is too smart to be safe: Stealthy chat with llms via cipher. *arXiv preprint arXiv:2308.06463*.