Multi-module GRPO: Composing Policy Gradients and Prompt Optimization for Language Model Programs

Noah Ziems[*1], **Dilara Soylu**[*2], **Lakshya A Agrawal**[*3], **Isaac Miller**[7], **Liheng Lai**[3], **Chen Qian**[5], **Kaiqiang Song**[8], **Meng Jiang**[1], **Dan Klein**[3], **Matei Zaharia**[3,5], **Karel D'Oosterlinck**[6], **Christopher Potts**[2], **Omar Khattab**[4]

[1]University of Notre Dame [2]Stanford University [3]UC Berkeley
[4]MIT [5]Databricks [6]Contextual AI [7]Normal Computing [8]Zoom, Inc.

Abstract

Group Relative Policy Optimization (GRPO) has proven to be an effective tool for post-training language models (LMs). However, AI systems are increasingly expressed as modular programs that mix together multiple LM calls with distinct prompt templates and other tools, and it is not clear how best to leverage GRPO to improve these systems. We begin to address this challenge by defining mmGRPO, a simple multi-module generalization of GRPO that groups LM calls by module across rollouts and handles variable-length and interrupted trajectories. We find that mmGRPO, composed with automatic prompt optimization via the BetterTogether method of Soylu et al. (2024), improves accuracy by 11% on average across classification, many-hop search, and privacy-preserving delegation tasks against the post-trained LM—and by 5% against prompt optimization on its own. We open-source mmGRPO as the `dspy.GRPO` optimizer in the DSPy library at `dspy.ai`.

https://github.com/stanfordnlp/dspy

```
program_po = MIPROv2(metric).compile(program, trainset)
program_rl = GRPO(metric).compile(program_po, trainset)
```

1 Introduction

Modern natural language processing (NLP) systems are increasingly implemented as modular systems, in which each module is responsible for a well-specified subtask that contributes to solving a broader objective. A canonical example is "multi-hop" research, where the system responds to a question by iteratively using a *query generation* LM module to produce a search query, passing that query to a retriever, and finally feeding all iteratively retrieved passages into a *response generation* LM module to produce the final output. The explicit modularization of such systems makes their behavior controllable, akin to conventional software, and allows for structured optimization of individual components, leveraging the priors of the LM differently for each module.

Group Relative Policy Optimization (GRPO; Shao et al. 2024) has recently emerged as a powerful method for fine-tuning language models (LMs) in the final stages of training. By leveraging relative rewards within groups of "reasoning" rollouts that share the same prompt, GRPO offers a simple

* Equal contribution.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Table 1: Performance of different learning algorithms across three LM programs: a single-stage program, `Banking77`, and multi-stage programs, `PAPILLON` and $\texttt{HoVer}_{\texttt{4-HOP}}$. MIPROv2 represents a prompt optimization baseline, while Vanilla CoT refers to vanilla chain-of-thought prompting. Both mmGRPO and MIPROv2 improve over the untuned baseline, though neither consistently dominates the other. The best overall performance is achieved by the BetterTogether variant of mmGRPO, which first applies prompt optimization using MIPROv2 and then fine-tunes using mmGRPO. We report dev set accuracy for each cell, averaged over 3 seeds. The dev set is used strictly for evaluation and not for optimization.

Strategy	Banking77		PAPILLON		$\texttt{HoVer}_{\texttt{4-HOP}}$		Avg Scores		
	llama3.1	qwen3	llama3.1	qwen3	llama3.1	qwen3	llama3.1	qwen3	All
Baseline Strategies:									
Vanilla CoT	58.4	64.6	76.2	78.3	59.5	60.6	64.7	67.8	66.3
MIPROv2 (PO)	59.4	65.9	83.9	78.1	63.4	69.3	68.9	71.1	70.0
mmGRPO *Strategies*:									
mmGRPO	**63.7**	64.9	83.9	**83.3**	60.2	71.0	69.3	73.1	71.2
BetterTogether(PO, mmGRPO)	**63.7**	**69.1**	**86.5**	81.1	**68.3**	**71.5**	**72.8**	**73.9**	**73.4**

alternative to Proximal Policy Optimization (PPO; Schulman et al. 2017). However, GRPO was originally designed for single-stage settings where each rollout consists of a single autoregressive LM call, and it is not obvious how to best extend it to systems composed of multiple such calls with distinct prompt templates.

In this paper, we ask whether post-training RL algorithms such as GRPO could be applied effectively to such multi-module LM programs, in which each rollout may invoke several distinct LM modules, each with its own prompt template and context. This could prove challenging in practice, as such the rollouts generated from the same input to the program can differ in both number of steps and structure, due to variations in control flow or early termination from, e.g., parsing failures, and often produce disjoint intermediate inputs and outputs. Generating rollouts with shared histories would require interfering with the control flow logic and, even then, collecting an exponential number of full trajectories to obtain samples that diverge at the same module call.

In response to these challenges, we implement mmGRPO, a simple and extensible framework for applying GRPO to multi-module setups. The core idea is to relax GRPO's requirement for shared inputs by grouping rollouts at the *module-level*, aligning structurally comparable module calls across different trajectories. This approach enables GRPO-style policy gradient updates without requiring shared histories or module-level inputs across rollouts, and it offers a first strong baseline for online policy-gradient RL methods applied to LM programs. We open-source mmGRPO as an off-the-shelf optimizer for arbitrary compound AI systems as part of the DSPy library at dspy.ai.

Ours is the first implementation of GRPO that applies to sophisticated pipelines of LMs. This enables us to conduct a controlled comparison of three approaches to optimizing modular AI systems: prompt optimization (PO), online reinforcement learning via mmGRPO, and their combination using the BetterTogether framework (Soylu et al., 2024). Our evaluation spans three diverse LM program tasks: classification (`Banking77`; Casanueva et al. 2020), multi-hop claim verification (`HoVer`; Jiang et al. 2020, and privacy-conscious delegation (`PAPILLON`; Siyan et al. 2024). Each involves different reasoning styles and control flow structures. Experiments are run using two open-source LMs, `llama3.1-8b-instruct` (Grattafiori et al., 2024) and `qwen3-8b` (Yang et al., 2025).

Our results are summarized in Table 1. Across these settings, mmGRPO improves performance by 7% on average against the model's unadapted reasoning performance. While mmGRPO does not always surpass the prompt optimized programs via MIPROv2 (Opsahl-Ong et al., 2024), it complements them effectively: staging MIPROv2 and mmGRPO—à la BetterTogether—consistently yields higher performance than either method alone, improving by 5% and 3% compared to MIPROv2 and mmGRPO, respectively; and by 11% compared to the model's unadapted reasoning performance. These findings suggest that policy gradient RL and PO offer complementary benefits for LM program training, and we advocate for future work exploring their integration in both offline and online settings.

2 Preliminaries

GRPO is an online policy gradient method for LM fine-tuning that operates over *groups* of trajectories sharing the *same input prompt* in *single-stage* tasks. The GRPO objective encourages the current policy $p_{\theta_{\text{old}}}$, parametrized by LM weights θ_{old}, to upweight relatively high-reward completions within a group, while applying PPO-style clipping and KL divergence regularization to ensure stable updates. This results in an updated policy p_θ.

GRPO also makes use of a reference policy $p_{\theta_{\text{ref}}}$ in the KL-divergence penalty, seeking to prevent the updated policy from drifting too far from its original distribution. Here, we express the original GRPO objective in Equation 1 in terms of the prompt–output–reward triples (q, o_i, r_i) to facilitate the extension to the multi-module setting.

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\{(q,o_i,r_i)\}_{i=1}^{G}}, \text{ where } \theta \text{ indicates the parameters for an LM shared by all groups}$$
$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left(\omega_t \hat{A}_i,\ \text{clip}\left(\omega_t,\ 1-\epsilon,\ 1+\epsilon\right)\hat{A}_i\right) - \beta\mathbb{D}_{\text{KL}}[p_\theta \,\|\, p_{\theta_{\text{ref}}}]\right\} \tag{1}$$
$$\text{where} \quad \omega_t = \frac{p_\theta(o_{i,t} \mid q, o_{i,<t})}{p_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}, \text{ and } \hat{A}_i \text{ is derived from the observed reward } r_i \text{ (below)}$$

Each GRPO group is defined as a set of triples $\mathcal{G} = \{(q, o_i, r_i)\}_{i=1}^{G}$, constructed by first sampling a fixed prompt from a distribution of questions $q \sim P(Q)$, and then generating a batch of G completions $\{o_i\}_{i=1}^{G} \sim p_{\theta_{\text{old}}}(O \mid q)$ from the current policy. Finally, a scalar reward r_i for each o_i is computed with a reward function. The term ω_t denotes the importance sampling ratio between the new and old policies for the tth token in a given output. The scalar reward r_i is then normalized within the group to compute an advantage $\hat{A}_i$ in the *outcome supervision* formulation of GRPO, which is applied uniformly across all tokens t in the corresponding completion, as shown in Equation 2.

$$\hat{A}_i = \frac{r_i - \text{mean}(\mathcal{R})}{\text{std}(\mathcal{R})}, \quad \mathcal{R} = \{r_i, \text{reward for } o_i\}_{i=1}^{G} \tag{2}$$

LM program formulation An LM program Φ is composed of LM modules and other tools orchestrated by the control flow of Φ. Let $\mathcal{M} = \{M_1, \ldots, M_{|\mathcal{M}|}\}$ denote the set of LM modules used therein, each of which communicates via natural language.

Given a structured input x (e.g., a record with fields such as `question`), executing $\Phi(x)$ orchestrates module invocations, transforming inputs and routing outputs between modules. In other words, $\Phi(x)$ defines a distribution from which we can sample y, ρ pairs, where y is the final output and ρ is the trajectory of module calls:

$$(y, \rho) \sim \Phi(x), \quad \rho = [\zeta_1, \zeta_2, \ldots, \zeta_{|\rho|}], \tag{3}$$

Here, the trajectory ρ records the sequence of module calls, and each trace $\zeta_t = \langle M_t, q_t, o_t \rangle$ captures the module identity as well as the module-level inputs and outputs at module invocation t within the program trajectory. The trajectory ρ logs only the LM-level calls in their execution order and omits any other control logic.

Each module $M \in \mathcal{M}$, which may appear zero or more times in a given ρ, is parameterized by a prompt template π_M and LM weights θ_M. During execution at module invocation t, the prompt template π_{M_t} transforms the input q_t into a materialized prompt: $q_t \leftarrow \pi_{M_t}(q_t)$. This prompt is then passed to an LM parameterized by θ_{M_t}, which samples an output $o_t \sim p_{\theta_{M_t}}(\cdot \mid q_t)$, returned to the control flow of Φ for subsequent steps.[1]

This modularity offers several benefits. It allows for privacy-preserving delegation, e.g., a module may call a proprietary LM that should not access previous interactions, as in our `PAPILLON` task, and better context length management, which is particularly important in RAG-style pipelines like `HoVer`,

[1]It is useful to consider how this setup differs from standard multi-turn LM generation settings, where the LM prompt is expanded serially in each turn (Jin et al. 2025; Zeng et al. 2025; Wang et al. 2025). In arbitrary LM programs, the control flow dictates what context is visible to each module by selecting its inputs, enabling more modular and interpretable execution, but presenting new challenges for learning.

where large numbers of retrieved passages may need to be processed independently. This is a core reason why multi-step GRPO formulations wouldn't be suitable for LM programs out-of-the-box and motivates us to explore alternative multi-module formulations. Throughout this paper, we treat LM policy inputs as being defined strictly at the module-level.

LM program optimization Let $\mathcal{D} = \{(x, m)\}$ be a dataset of inputs x and optional metadata m (e.g., final answer, documents to retrieve, or PII to redact). The goal is to learn the parameters of the given LM program Φ, namely, the prompt templates π_M and LM weights θ_M for each module $M \in \mathcal{M}$, such that we maximize the expected reward $\mathbb{E}_{(x,m)\sim\mathcal{D};\,(y,\rho)\sim\Phi_{\Pi,\Theta}(x)}[\mu(y, \rho, m)]$.

Here, the reward function $\mu(y, \rho, m)$ scores the execution, typically based on the final output y's correctness. Any metadata m (e.g., gold answers) is not visible to the program during execution but may be used by μ for evaluation.

3 Applying GRPO to multi-module LM programs

Given a dataset $\mathcal{D}$ and a reward function μ, our goal is to optimize an LM program Φ consisting of modules $\mathcal{M}$ by updating the weights θ_{M_i} of each module. In standard GRPO, each group contains trajectories from a single auto-regressive LM call—i.e., one prompt and its full output. LM programs typically comprise multiple modules, each invoking its own LM with a custom prompt, raising the question of how to best extend GRPO grouping to this multi-module setting. To set a strong baseline in this space, we explore the simplest possible design with mmGRPO, particularly one that allows our implementation to remain largely modular with respect to existing GRPO implementations.

mmGRPO starts by sampling full program trajectories, forming a meta-group of trajectories, each with many module invocations. It then aligns module calls across these trajectories and creates GRPO groups at the module level, each containing input–output–reward triples for a specific module. We default to uniform credit assignment, setting each reward to correspond to the final program reward. A modified GRPO loss is then applied independently to each group, updating only the LM weights of the module that produced the group's data. In practice, the same LM is often shared across all modules. Section 5 validates that this approach is able to improve realistic LM programs and to compose effectively with prompt optimization. We focus on the high-level design in this section, deferring implementation details to Appendix A.

Additionally, mmGRPO allows sampling trajectories not only from the student program but also from a list of fixed *teacher* programs. This enables flexible training setups, including warm-starting from prompt-optimized programs or learning from more capable LMs. When used on single-module programs without teachers, mmGRPO reduces exactly to standard GRPO.

The meta-group of trajectories used in mmGRPO consists of multiple executions of the same program on a shared program-level input x, i.e., $(y, \rho) \sim \Phi(x)$, where y is the final program output and $\rho = [\zeta_1, \zeta_2, \ldots, \zeta_{|\rho|}]$ is the trajectory of module calls. Each ζ_t is a triple containing the invoked module M_t, the prompt q_t sent to the corresponding module LM θ_{M_t}, and the resulting output o_t. The program-level output reward for the entire trajectory is computed as $r = \mu(y, \rho, m)$, where m is any additional metadata associated with the example.

To construct GRPO groups, mmGRPO aligns module calls across trajectories based on both the module identifier and the relative order in which it appears within the trajectory. This alignment process yields module-level GRPO groups, each of the form $\{(q_i, o_i, r_i)\}_{i=1}^{G}$, where q_i and o_i are extracted from a group of aligned traces all generated by a specific module M, and r_i is set to the corresponding program-level output reward for the trajectory that generated each trace.

$$\mathcal{J}_{\text{mmGRPO}}(\theta_M) = \mathbb{E}_{\{(q_i, o_i, r_i)\}_{i=1}^{G}}, \text{ where } \theta_M \text{ indicates the LM weights for module } M$$

$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left(\omega_t\hat{A}_i,\ \text{clip}\left(\omega_t,\ 1-\epsilon,\ 1+\epsilon\right)\hat{A}_i\right) - \beta\mathbb{D}_{\text{KL}}[p_{\theta_M} \,\|\, p_{\theta_{M_{\text{ref}}}}]\right\} \tag{4}$$

$$\text{where} \quad \omega_t = \frac{p_{\theta_M}(o_{i,t} \mid q_i, o_{i,<t})}{p_{\theta_{M_{\text{old}}}}(o_{i,t} \mid q_i, o_{i,<t})}, \text{ and } \hat{A}_i \text{ is computed from } r_i \text{ via Equation 2}$$

In practice, not all trajectories generated by Φ given the same program-level input x follow the same structure; the program logic may diverge (e.g., by invoking different modules or terminating early), or errors such as module-level parsing failures may halt execution. To accommodate this, mmGRPO optionally pads smaller groups to a fixed size before applying the loss, described in more detail in Appendix A. Once the groups are formed, mmGRPO loss in Equation 4 is applied independently to each module-level group, with two key differences from the original GRPO objective (Equation 1). First, rather than updating a shared LM, each group updates only the weights of the module it corresponds to. Second, unlike GRPO where completions share a single prompt, datapoints in a module-level group may have different prompts q_i, reflecting variation in upstream context.

4 Composing Online RL with Prompt Optimization via BetterTogether

BetterTogether (Soylu et al., 2024) demonstrated that combining prompt optimization (PO) with weight optimization yields stronger results than using either technique alone, specifically in the context of offline RL via rejection fine-tuning on outcome-filtered trajectories. Rather than applying weight optimization directly to an unmodified program, the authors first optimize the program's prompt templates and then apply weight optimization on the resulting prompt-optimized program.

We extend this approach to the online RL setting using mmGRPO, and combine it with a state-of-the-art prompt optimizer, MIPROv2 (Opsahl-Ong et al., 2024). Soylu et al. (2024) also experiment with alternative compositions, such as running prompt optimization after weight tuning, but in our work, we focus on the former: applying mmGRPO to a prompt-optimized program.

5 Experiments

5.1 LMs and datasets

The LM programs for each of the tasks we use for evaluation, along with example inputs and program trajectories, are shared in Appendix B. We use the LM program implementations open-sourced by Tan et al. (2025) as our starting point for all tasks, but make modifications for `HoVer`. For more information on the LMs and datasets used along with their license information, refer to Appendix C.

LMs We run our experiments on two open LMs: `llama3.1-8b-instruct` (Grattafiori et al., 2024) and `qwen3-8b` (Yang et al., 2025). Although mmGRPO allows for different LM copies to learn separate weight updates for the different modules of a program, we use the same underlying LM weights for each module for lightweight training and deployment in a multi-task manner.

Classification `Banking77` is an intent classification benchmark involving $13,083$ labeled customer service queries from the banking domain Casanueva et al. (2020). The task is to assign each user query to one of 77 intent classes. We implement a simple program for this task using a single Chain-of-Thought (CoT) module (Wei et al., 2022), which first produces a reasoning trace before predicting the intent label. For evaluation we compute the exact match between the ground-truth label and the generated label. Since the program we have for `Banking77` has only a single module, running the mmGRPO algorithm on it is the same as the standard GRPO setup. For training and evaluation, we randomly sample 250 training examples and 500 for development.

Privacy-conscious delegation The Private User Prompt Annotations (PUPA) benchmark constructed by Siyan et al. (2024) focuses on privacy-preserving question answering, where the goal is to respond to user queries without exposing private information to external APIs. We use PAPILLON, also from Siyan et al. (2024), a two-module pipeline that generates a redacted version of a private user query, sends the redacted query to an untrusted but more powerful external model, and then uses the response of that powerful model to generate the final response. We utilize `openai/gpt-4.1-mini-2025-04-14` (OpenAI, 2025) as the external LM. As described in Siyan et al. (2024), the evaluation metric is a composite score which takes into account the content of the response and the amount of private information that was leaked, both of which are judged by the same large LM. We evaluate this setup using 111 training examples and 221 for development.

Multi-hop claim verification `HoVer` (Hoppy Verification, Jiang et al., 2020) is a claim verification benchmark where the task is to extract facts from multiple relevant Wikipedia articles and deciding whether a given claim is supported. The claims in `HoVer` are *multi-hop* in that they require multi-hop reasoning by connecting information found in different articles. The original dataset has $18,171$ train and 4000 development and test examples derived from the examples in the HotPotQA dataset (Yang et al., 2018). Our program for `HoVer` consists of 2 modules, a query generation module and a fact summarization module, called iteratively over 4 hops, along with a ColBERTv2 (Santhanam et al., 2021) retriever indexed on the short snippets from the Wikipedia (2017) dump provided with the HotPotQA dataset, shared with `HoVer`. We refer to the particular 4-hop variant Hover program we use with $\texttt{HoVer}_{\texttt{4-HOP}}$, in order to differentiate it from the one provided in Tan et al. (2024). The program returns up to 100 passages at the end, and the final metric evaluates whether the gold passages are found within the returned passages using Recall@100. We build our splits from the original train split, randomly sampling 500 examples each for our train and development splits; while ensuring that we don't sample any two examples derived from the same HotPotQA question.

5.2 Baseline and method details

We evaluate each of our LM and task pairs with vanilla Chain-of-Thought (CoT) and a prompt optimizer, to serve as baselines. We demonstrate our MMGRPO optimizer in two flavors: MMGRPO, and BetterTogether MMGRPO. While each method assumes access to a program-level evaluation metric, none relies on an external oracle dataset. Instead, we generate training data dynamically by running the program itself and bootstrapping from model outputs and associated program-level metrics. We use the DSPy framework (Khattab et al., 2024) to run our baseline experiments and develop our new MMGRPO optimizers. We use DSPy's RL training library, Arbor (Ziems et al., 2025), which draws inspiration from the Verifiers library (Brown, 2025).

Inference We use the `vLLM` (Kwon et al., 2023) engine for sampling with max context length of $32,768$ tokens for inference. We set max tokens to 1032 and re-try each query up to 3 times in case of module parsing errors. For `qwen3-8b`, we use $\text{sampling\_temperature} = 0.6$, $\text{top\_p} = 0.95$ and $\text{top\_k} = 20$ following the parameters used for its instruction training as noted in Yang et al. (2025). For `llama3.1-8b-instruct`, we use $\text{sampling\_temperature} = 0.6$ and $\text{top\_p} = 0.9$ following the official model card's generation configuration in HuggingFace (MetaAI, 2024).

Vanilla CoT We adopt the Chain-of-Thought (CoT) prompting method introduced by Wei et al. (2022), where each module's prompt instructs the language model to first generate a *reasoning* field before producing its final answer. Unless stated otherwise, both the prompt-optimization and MMGRPO methods described below begin training from this base CoT prompt. We refer to this initial prompt configuration as the "Vanilla CoT" program.

MIPROv2 We use the state-of-the-art prompt optimizer Multiprompt Instruction PRoposal Optimizer Version 2 (MIPROv2; Opsahl-Ong et al. 2024) as our prompt-optimized baseline. For our experiments, we use the `auto=medium` setting, which uses `12` trials; `12` few-shot and `6` instruction candidates, and automatically uses a 80% of the train set for validation. We refer to the program we optimize using MIPROv2 with these settings as the prompt-optimized program and re-use it for the BetterTogether strategy below.

mmGRPO We train our models using the HuggingFace `GRPOTrainer`, each with a maximum context length of 8192 tokens. Training is performed with a temperature of 0.6, a learning rate of 1×10^{-5}, gradient accumulation steps of 20, with per device train batch size of 1. We use $\beta = 0.01$ and gradient norm clipping of 0.1 for `qwen3-8b`; and $\beta = 0.04$ and gradient norm clipping of 0.5 for `llama3.1-8b-instruct`.

We run MMGRPO for 750 steps, using 4 training examples per step. At each step, we randomly draw 4 examples from the training dataset. For each example, we generate 12 rollouts, which are then grouped into module-level GRPO groups using the procedure in Algorithm 2. We use a train context length of $8,192$ tokens, which is used to filter any trajectory with a module level prompt and completion longer than this. We apply Low-Rank Adaptation (LoRA, Hu et al. 2021) with rank $\text{r} = 16$, $\text{lora\_alpha} = 64$, $\text{lora\_dropout} = 0.05$, targeting the projection modules [q, k, v, o, up, down, gate]. We run all of our mmGRPO experiments below using these same settings. Pseudocode of the mmGRPO algorithm can be found in Algorithm 1.

mmGRPO with BetterTogether We further experiment with a setting where we combine prompt optimization with the weight optimization of mmGRPO following the BetterTogether algorithm introduced by Soylu et al. (2024). Specifically, instead of directly optimizing the weights used in an LM program, we first use prompt optimization to find high quality prompts to be used by the LM program. The prompts are then kept fixed in the LM program and the program weights are then optimized with mmGRPO. We refer to this setup as BetterTogether(PO, mmGRPO) for short.

5.3 Main results

Our main experimental results are shared in Table 1, evaluated on the dev set and averaged over 3 seeds. The dev set is used exclusively for evaluation and plays no role in optimization. [2]

mmGRPO and BetterTogether(PO, mmGRPO) consistently improve over their respective baselines. We can see that the mmGRPO row is consistently higher than the "Vanilla CoT" row, 7% on average. Similarly, BetterTogether(PO, mmGRPO) shows consistent gains over the "MIPROv2 (PO)" row, 5% on average. These show that mmGRPO is effective at finding better policies for the provided program across all LM–task pairs.

PO is competitive with lower computational budgets. When averaged across all tasks and models, MIPROv2 alone improved upon the Vanilla CoT strategy by 5% compared to mmGRPO's 7% improvement. However, MIPROv2 achieved these results significantly faster while using fewer GPU-hours. On average, our vanilla mmGRPO experiments took 18.7 hours using 2 `H100` GPUs whereas MIPROv2 took only 1.4 hours on average and only required 1 `H100` GPU. These results indicate that PO approaches like MIPROv2 are likely much more feasible for settings which have lower computation budgets.

BetterTogether(PO, mmGRPO) performs the best in most task pairs. BetterTogether(PO, mm-GRPO) approach improves over the Vanilla CoT by 11%, MIPROv2 by 5%, and vanilla mmGRPO by 3%. This shows the value of high-quality rollouts at the start of mmGRPO training, as performing PO generates stronger rollouts, leading to a more robust training signal early in the training runs.

6 Related work

Prompt optimization Much recent work has explored methods that adapt prompt strings to fit data. This includes methods focused on prompting LMs to generate instructions (Yang et al., 2024; Zhou et al., 2023; Pryzant et al., 2023; Fernando et al., 2024), using gradients to optimize the prompt (Shin et al., 2020; Wen et al., 2023), and RL-based prompt optimizers (Deng et al., 2022; Zhang et al., 2023; Hao et al., 2023), among many others.

Weight optimization Proximal Policy Optimization (PPO) has been widely used for post-training language models with reinforcement learning, particularly when aligning language models with human preferences or feedback (Schulman et al., 2017; Ouyang et al., 2022). Recently, Direct Preference Optimization (DPO) algorithms emerged as a simpler alternative that avoids explicit reward modeling and instead learns from contrastive preference pairs (Rafailov et al., 2023). Similarly, Group Relative Policy Optimization (GRPO) offers an efficient alternative to PPO by avoiding the need for a value model and instead relying on estimated advantages through relative rewards within a group of rollouts (Shao et al., 2024).

Optimization of LM Programs' Prompts & Weights Existing work has explored optimizing LM programs with prompt optimizers, including those that focus primarily on rejection sampling (Khattab et al., 2024) and others that extend this to use Bayesian optimization for selecting the instruction-demonstration candidates that are most promising (Opsahl-Ong et al., 2024). Additional

[2] Instructions and the code for running the experiments in the paper can be found at https://github.com/dilarasoylu/mmgrpo

work (Soylu et al., 2024) has explored combining weight optimizers with prompt optimizers for additional benefit, but in the context of offline RL. However, adapting some techniques to LM Programs requires making a number of decisions (Section 2) and presents substantial implementation challenges. The present work describes how we generalize GRPO to LM programs composed of multiple modules.

7 Conclusion

We introduce mmGRPO, a novel extension of GRPO that enables online weight optimization for multi-module LM programs by propagating final rewards backward across disjoint modules. Our experiments demonstrate that mmGRPO consistently outperforms standard baselines across tasks and models, validating its effectiveness in navigating the challenging credit assignment problem without requiring intermediate supervision. We further show that combining mmGRPO with state-of-the-art prompt optimization methods via BetterTogether yields the strongest overall performance in the majority of settings, revealing that complementary relationship between weight and prompt optimization holds for online RL methods.

8 Limitations

While our experiments demonstrate the promise of multi-module RL formulations, this work has several limitations. First, we use 8-billion parameter language models, which may not reflect how mmGRPO performs with larger models. Second, we rely on LoRA for fine-tuning; while efficient, this may limit training performance compared to full-parameter updates. Third, we evaluate only one mmGRPO implementation despite many possible alternative formulations. Finally, while `Banking77` is a well-understood classification task, we study it in a limited-feedback setting where models only receive rewards derived from bootstrapped rollouts, not supervised intent labels. While supervised training enables encoder models to perform well on this task, we investigate whether GRPO or MIPRO can achieve similar performance from reward signals alone. Our results suggest that this is not yet the case.

Acknowledgments

D.S. thanks the members of the StanfordNLP group for their support and feedback. The authors also thank Zoom, Inc., for providing access to their compute clusters for running the later batch of experiments.

References

William Brown. Verifiers: Reinforcement learning with llms in verifiable environments. https://github.com/willccbb/verifiers, 2025.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pp. 38–45, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.nlp4convai-1.5. URL https://aclanthology.org/2020.nlp4convai-1.5.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3369–3391, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.222. URL https://aclanthology.org/2022.emnlp-main.222/.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 13481–13544, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yaru Hao, Zewen Chi, Li Dong, and Furu Wei. Optimizing prompts for text-to-image generation. *Advances in Neural Information Processing Systems*, 36:66923–66939, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ArXiv preprint*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. HoVer: A dataset for many-hop fact extraction and claim verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3441–3460, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.309. URL https://aclanthology.org/2020.findings-emnlp.309.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *ArXiv preprint*, abs/2503.09516, 2025. URL https://arxiv.org/abs/2503.09516.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. In *ICLR*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

MetaAI. Meta llama 3.1-8b-instruct: Generation configuration. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct/blob/main/generation_config.json, 2024. Accessed: 2025-07-29.

OpenAI. Gpt-4.1-mini. https://platform.openai.com/docs/models/gpt-4.1-mini, 2025. Accessed: 2025-07-17.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.525. URL https://aclanthology.org/2024.emnlp-main.525/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *ArXiv preprint*, abs/2112.01488, 2021. URL https://arxiv.org/abs/2112.01488.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv preprint*, abs/1707.06347, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv preprint*, abs/2402.03300, 2024. URL https://arxiv.org/abs/2402.03300.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.346. URL https://aclanthology.org/2020.emnlp-main.346.

Li Siyan, Vethavikashini Chithrra Raghuram, Omar Khattab, Julia Hirschberg, and Zhou Yu. Papillon: Privacy preservation from internet-based and local language model ensembles. *ArXiv preprint*, abs/2410.17127, 2024. URL https://arxiv.org/abs/2410.17127.

Dilara Soylu, Christopher Potts, and Omar Khattab. Fine-tuning and prompt optimization: Two great steps that work better together. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10696–10710, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.597. URL https://aclanthology.org/2024.emnlp-main.597/.

Shangyin Tan, Lakshya A Agrawal, Arnav Singhvi, Liheng Lai, Michael J Ryan, Dan Klein, Omar Khattab, Koushik Sen, and Matei Zaharia. Langprobe: a language programs benchmark. *ArXiv preprint*, abs/2502.20315, 2025. URL https://arxiv.org/abs/2502.20315.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. Judgebench: A benchmark for evaluating llm-based judges. *ArXiv preprint*, abs/2410.12784, 2024. URL https://arxiv.org/abs/2410.12784.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *ArXiv preprint*, abs/2504.20073, 2025. URL https://arxiv.org/abs/2504.20073.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *ArXiv preprint*, abs/2201.11903, 2022. URL https://arxiv.org/abs/2201.11903.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36:51008–51025, 2023.

Yixuan Even Xu, Yash Savani, Fei Fang, and Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *ArXiv preprint*, abs/2504.13818, 2025. URL https://arxiv.org/abs/2504.13818.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *ArXiv preprint*, abs/2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. *ArXiv preprint*, abs/2505.11821, 2025. URL https://arxiv.org/abs/2505.11821.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. TEMPERA: Test-time prompt editing via reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=gSHyqBijPFO.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

Noah Ziems, Lakshya A Agrawal, Dilara Soylu, Liheng Lai, Isaac Miller, Chen Qian, Meng Jiang, and Omar Khattab. Arbor: Open source language model post training. https://github.com/Ziems/arbor, 2025.

Appendix

A MMGRPO algorithm

A.1 Overview

The MMGRPO algorithm extends GRPO to the multi-module setting by improving the LM weights of each module in a program through module-level policy gradients. Two core abstractions distinguish MMGRPO in Algorithm 1: (1) the ability to sample trajectories from multiple teacher programs, and (2) the construction of module-level GRPO groups based on relative invocation order. These components are highlighted in the algorithm and explained in more detail in Section A.2 and Section A.3, respectively, while the remaining steps follow standard GRPO procedure and are included for completeness.

Algorithm 1 MMGRPO: GRPO for multi-module LM programs

Require:

Student program Φ, with modules $M \in \mathcal{M}$
Training set $\mathcal{D}$
Metric μ
Teacher programs $\mathcal{T}$ (optional), defaults to a list containing only the student program if left empty
Data collection hyper-parameters Ψ_{data} (optional):
  number of training steps N_{steps}
  batch size B
  rollout configuration $K : \mathcal{T} \to \mathbb{N}$, specifying the number of rollouts per example for each teacher
Model training hyper-parameters Ψ_{train} (optional): learning rate η, weight decay λ, and others
Shared hyper-parameters Ψ_{shared} (optional): group size G

1: **function** MMGRPO(Φ, $\mathcal{D}$, μ, $\mathcal{T}$, Ψ_{data}, Ψ_{train}, Ψ_{shared})
2:   **for** step $= 1$ to N_{steps} **do**
3:     $\mathcal{B} = \textsc{SampleBatch}(\mathcal{D}, B)$
4:     **for** $(x, m) \in \mathcal{B}$ **do**
5:       $\mathcal{R} \leftarrow \textsc{SampleTeacherRollouts}(\mathcal{T}, K)$
6:       `grpo_groups`, $\Theta \leftarrow \textsc{FormModuleLevelGroups}(\Phi, \mathcal{R}, G, \mu, x, m)$
7:       **for** each group $\mathcal{G} \in$ `grpo_groups` and corresponding module LM weights $\theta_M \in \Theta$ **do**
8:         Update θ_M via the GRPO objective in Equation 4 using hyper-parameters $\Psi_{\text{train}} \cup \Psi_{\text{shared}}$
9:   **return** Φ with the same prompt-templates but improved LM weights, i.e., $\{\pi_{M_i}, \theta^*_{M_i}\}_{i=1}^{|\mathcal{M}|}$
10:
11: **function** SAMPLETEACHERROLLOUTS($\mathcal{T}$, K, x, m)
12:   $\mathcal{R} \leftarrow \emptyset$
13:   **for** each teacher program $\Phi^{(t)} \in \mathcal{T}$ **do**
14:     `num_samples` $\leftarrow K[\Phi^{(t)}]$
15:     **for** $k = 1$ to `num_samples` **do**
16:       $(y, \rho) \sim \Phi^{(t)}(x)$
17:       $\mathcal{R} \leftarrow \mathcal{R} \cup \{(y, \rho)\}$
18:   **return** $\mathcal{R}$

Assume SAMPLEBATCH is provided
Refer to Algorithm 2 for FORMMODULELEVELGROUPS

MMGRPO takes as input a student program Φ, a training dataset $\mathcal{D}$, a reward metric μ, an optional set of teacher programs $\mathcal{T}$, and optional hyper-parameters (Line 1). If unspecified, the set of teacher programs $\mathcal{T}$ defaults to a singleton set containing only the student program. At each training step (Line 2), the algorithm samples a batch $\mathcal{B}$ of examples from the training dataset $\mathcal{D}$ using the configured batch size B (Line 3). For each example $(x, m) \in \mathcal{B}$ (Line 4), the algorithm collects rollouts from the teacher programs via the SAMPLETEACHERROLLOUTS function (Line 5), which returns a set of output-trajectory tuples. These rollouts are passed to FORMMODULELEVELGROUPS from Algorithm 2 (Line 6), which constructs module-level GRPO groups and returns them along with the corresponding references to the module-level LM weights θ_M to be updated. The algorithm then iterates over each group and its associated LM weights (Line 7), and applies the GRPO loss (as defined in Equation 4) independently to each group (Line 8), using the specified training hyper-parameters. After N_{steps} iterations, the algorithm returns the updated student program Φ, preserving its original prompt templates while incorporating improved LM weights (Line 9).

A.2 Sampling with teacher programs

In addition to the student program, MMGRPO accepts a list of optional *teacher programs*, which are used to generate the set of trajectories that populate the `runs` list. At each GRPO step, rather than sampling all rollouts from the student program alone, MMGRPO samples trajectories from a specified mixture of teacher programs. This list must include the student itself. All teacher programs share the same structural interface, meaning they operate over the same LM program and module-level input/output fields, but may differ in their module-level prompt-templates (e.g., alternative instructions or few-shot examples) or LM weights (e.g., larger LMs). These variations enable the MMGRPO framework to support training that is online but partially off-policy, providing greater flexibility in guiding learning using curated or higher-performing policies.

The SAMPLETEACHERROLLOUTS function samples trajectories from each teacher program in $\mathcal{T}$, using a rollout configuration K that specifies the number of rollouts to generate per teacher. This per-teacher control enables flexible data mixtures across programs. For each rollout, the function extracts the final output y and trajectory ρ, and collects the resulting (y, ρ) pairs into the rollout set $\mathcal{R}$ returned for training.[3]

A.3 Forming module-level groups

Algorithm 2 FORMMODULELEVELGROUPS: Create module-level GRPO groups for MMGRPO

Require:

Student program Φ, with modules $M \in \mathcal{M}$
Rollouts $\mathcal{R} = \{(y_j, \rho_j)\}_{j=1}^{R}$, sampled outputs along with their trajectories
Group size G
Metric μ
Input x
Input metadata m

function FORMMODULELEVELGROUPS(Φ, $\mathcal{R}$, G, μ, x, m)
  `grpo_groups_dict` $\leftarrow$ DEFAULTDICT(list)
  for each $(y, \rho) \in \mathcal{R}$ **do**
    $r = \mu(y, \rho, m)$
    `relative_invocation_orders` $\leftarrow$ DEFAULTDICT(LIST)
    for each trace $\zeta = (M, q, o) \in \rho$ **do**
      Append (q, o, r) to `grpo_groups`[(M, `relative_invocation_orders`[M])]
      `relative_invocation_orders`[M] += 1
  `grpo_groups_dict` $\leftarrow$ PADGROUPS(`grpo_groups`)
  `grpo_groups` $\leftarrow$ [SELECTKDIVERSEELEMENTS($\mathcal{G}, G$) | $\mathcal{G} \in$ VALUES(`grpo_groups_dict`)]
  $\Theta \leftarrow$ [Get M's weights θ_M | (M, `relative_invocation_order`) $\in$ KEYS(`grpo_groups_dict`)]
  return `grpo_groups`, Θ

Assume DEFAULTDICT, KEYS, and VALUES are provided
Refer to Section A.3 for descriptions of PADGROUPS and SELECTKDIVERSEELEMENTS

We now describe how MMGRPO constructs GRPO-style groups at the module level for LM programs. Once the rollouts are sampled , MMGRPO construct *module-level* GRPO groups via the FORMMODULELEVELGROUPS function described in Algorithm 2. Each GRPO group is defined as a list of $G \leq R$ triples $\{(q_i, o_i, r_i)\}_{i=1}^{G}$, where each element consists of a module-level input prompt q, the corresponding output o, and the final trajectory-level reward r. In practice, one can use $G < R$, the number of rollouts, to leave room for post-hoc adjustments to group size (discussed later in this section).

[3] When using teacher programs to sample trajectories, the modules M recorded in the traces reflect those of the teacher rather than the student program. In practice, however, MMGRPO ensures that the module keys used to form module-level GRPO groups correspond to the student program's modules for each respective teacher module, since it is required that student and teachers programs share the "same structure".

Given the program Φ, the list of output–trajectory tuples $\mathcal{R}$, and the desired GRPO group size G, FormModuleLevelGroups iterates over each output–trajectory pair in $\mathcal{R}$ (Line 3), computing a corresponding score $r = \mu(y, \rho, m)$ (Line 4). If the corresponding trajectory is incomplete, a fallback reward is assigned (e.g., a formatting error penalty). Following this, it iterates over the traces in each trajectory (Line 6). Each trace contributes a triple (q, o, r) consisting of the module-level input, output, and final trajectory reward. This triple is added to the group corresponding to (M, k), where k is the relative invocation index of M in the trajectory (Line 7), where the relative index is incremented after each occurrence (Line 8). To ensure uniform group sizes despite variability in module invocation counts across trajectories, Lines 9 and 10 apply post-processing steps that adjust each group to have exactly G elements, as detailed later in this section. Finally, Line 11 constructs a list of LM weight references, one corresponding to each group, and both this list and the final GRPO groups are returned (Line 12).

As a result, FormModuleLevelGroups creates GRPO groups by both the module identity and their relative position within the trajectory with respect to the other calls to the same module. Let K_{M_i,ρ_j} denote the number of times module M_i is invoked in trajectory ρ_j for $(y_j, \rho_j) \in \mathcal{R}$; then the total number of GRPO groups formed across all trajectories is $\sum_i \max_j K_{M_i,\rho_j}$, where $M_i \in \mathcal{M}$ for the given `runs`. Each resulting group is a list of module-level (q, o, r) triples, corresponding to structurally aligned invocations of a given module at a specific position in the trajectory. In contrast to standard GRPO, which produces a single group per set of rollouts in single-stage settings, mmGRPO yields a list of groups, one for each module and relative invocation position. To ensure uniform group sizes and handle variation across trajectories, mmGRPO apply two *post-processing* steps: PadGroups and SelectKDiverseElements, described next.

Handling variably invoked trajectories with PadGroups If every module M_i in the student program is invoked the same number of times $K_{M_i,*}$ across all trajectories ρ_j where $(y_j, \rho_j) \in \mathcal{R}$, then each constructed GRPO group will contain exactly R triples prior to the call to Line 9 in Algorithm 2. For example, suppose the LM program consists of two modules, M_1 and M_2, and $R = 3$ trajectories are sampled. If, in every trajectory, the program calls M_1 exactly twice and M_2 exactly once, then mmGRPO will form three GRPO groups: two for M_1 (corresponding to its first and second calls) and one for M_2. Each of these groups will contain exactly three triples, one from each trajectory, without requiring any padding or truncation. This scenario arises when all executions yield structurally identical trajectories and none encounter parsing or runtime errors.

However, in practice, these conditions may not hold: some modules may be invoked fewer times due to variation in control flow, while others may terminate early due to parsing failures or other runtime errors. In such cases, certain module, module invocation level GRPO groups may contain fewer than N elements. To address this, mmGRPO applies post-processing strategies to ensure that each group has a uniform size, with a call to the PadGroups function, described here.

The behavior of PadGroups is controlled by a `padding_mode` hyper-parameter (not explicitly noted in the function call to it in Algorithm 1), which supports two values: `truncate` and `fill`. Under the `truncate` strategy, it discards all GRPO groups for module M_i whose invocation index exceeds $\min_j K_{M_i,\rho_j}$, ensuring that only groups with complete representation across all trajectories are retained. Under the `truncate` strategy, it discards all GRPO groups for a module M_i whose invocation index exceeds $\min_j K_{M_i,\rho_j}$, ensuring that only those invocation positions represented in every trajectory are retained. We use the `fill` setting for the experiments reported in this paper.

Ensuring diversity in groups with SelectKDiverseElements After standardizing group sizes across trajectories, mmGRPO further adjust seach group to ensure it contains exactly G elements, the target GRPO group size. Rather than sampling elements uniformly at random,it invokes the SelectKDiverseElements function, which selects (or duplicates) elements to form a group of size G while maximizing diversity within the group. This function handles both down-sampling (when the group has more than G elements) and up-sampling (when it has fewer), favoring selections that increase reward variance in the sampled prompt-output pairs. Contemporaneously, Xu et al. (2025) propose a similar variance-based selection strategy, demonstrating that promoting diversity in GRPO groups improves held-out generalization.

B Task details

The DSPy implementations for the LM programs for `Banking77`, `PAPILLON`, and `HoVer` are presented alongside example datapoints in their respective sections. Code snippets assume respective DSPy imports are available.

B.1 Banking77

The following snippet presents an example datapoint from `Banking77`. The task is to predict the appropriate label given a user query.

Example datapoint for Banking77

text: I have been waiting over a week. Is the card still coming?

label: card_arrival

A DSPy program for `Banking77` is shared in Python Snippet 1.

```python
LABELS = [ ... ]  # List of all output labels
Banking77 = dspy.ChainOfThought(f"text -> label: Literal{LABELS}")
```

Python Snippet 1: DSPy program for Banking77.

B.2 PAPILLON

The following snippet shows an example datapoint from `PAPILLON`. The task is to generate the target response using a proprietary language model, while ensuring that no personally identifiable information (PII) is leaked to it.

Example datapoint for PAPILLON

user_query: Send a friendly e-mail opening to Sara telling her thank you for calling me back so quickly, taking the time to discuss with me FRC's IPAEP, and for your immediate willingness to make reasonable accommodations to help break down the barriers caused by my disabilities so that I can have equal access to your program. Your kindness and compassion is appreciated.

target_response: Dear Sara, I hope this email finds you well. I just wanted to take a moment to express my sincere gratitude for your prompt response to my call and the time you took to discuss FRC's IPAEP with me. I truly appreciate your immediate willingness to make reasonable accommodations to ensure that I have equal access to your program and to help break down the barriers caused by my disabilities...

pii_str: sara||ipaep||frc

A DSPy program for `PAPILLON` is shared in Python Snippet 2.

```python
class CraftRedactedRequest(dspy.Signature):
    """
    Given a private user query, create a privacy-preserving
        request for a powerful external LLM.
    The LLM may assist without learning private information about
        the user.
    """

    user_query = dspy.InputField()
    llm_request = dspy.OutputField()


class RespondToQuery(dspy.Signature):
    """
    Respond to a user query.
```

```
    For inspiration, we found a potentially related request to a
        powerful external LLM and its response.
    """

    related_llm_request = dspy.InputField()
    related_llm_response = dspy.InputField(desc="information from
        a powerful LLM responding to a related request")
    user_query = dspy.InputField(desc="the user's request you need
         to fulfill")
    response = dspy.OutputField(desc="your final response to the
        user's request")


class PAPILLON(dspy.Module):
    def __init__(self, untrusted_model):
        self.craft_redacted_request = dspy.ChainOfThought(
            CraftRedactedRequest)
        self.respond_to_query = dspy.Predict(RespondToQuery)
        self.untrusted_model = untrusted_model

    def forward(self, user_query):
        llm_request = self.craft_redacted_request(user_query=
            user_query).llm_request
        llm_response = self.untrusted_model(llm_request)[0]
        response = self.respond_to_query(
            related_llm_request=llm_request, related_llm_response=
                llm_response, user_query=user_query
        ).response

        return dspy.Prediction(llm_request=llm_request,
            llm_response=llm_response, response=response)
```

Python Snippet 2: DSPy program for Papillon.

B.3 HoVer

The following snippet shows an example datapoint from HoVer. The task is to retrieve all gold Wikipedia titles that support the given claim.

Example datapoint for HoVer

claim: This director is known for his work on Miss Potter. The Academy of Motion Picture Arts and Sciences presents the award in which he was nominated for his work in "Babe".

titles: ['Miss Potter', 'Chris Noonan', 'Academy Award for Best Director']

A DSPy program for HoVer is shared in Python Snippet 3.

```
# Assume that a function called deduplicate is defined

class GenerateThreeQueries(dspy.Signature):
    """
    Given a claim and some key facts, generate up to 3 followup
        search query to find the next most essential clue towards
        verifying or refuting the claim. If you think fewer
        queries are sufficient, generate None for the search query
         outputs you don't need. The goal ultimately is to find
        all documents implicated by the claim.
    """
    claim = dspy.InputField()
    key_facts = dspy.InputField()
    search_query1 = dspy.OutputField()
```

```python
    search_query2 = dspy.OutputField()
    search_query3 = dspy.OutputField()


class AppendNotes(dspy.Signature):
    """
    Given a claim, some key facts, and new search results,
        identify any new learnings from the new search results,
        which will extend the key facts known so far about the
        whether the claim is true or false. The goal is to
        ultimately collect all facts that would help us find all
        documents implicated by the claim.
    """
    claim = dspy.InputField()
    key_facts = dspy.InputField()
    new_search_results = dspy.InputField()
    new_key_facts = dspy.OutputField()


class Hover(dspy.Module):
    def __init__(
            self,
            num_hops=4,
            k_per_search_query=10,
            k_per_search_query_last_hop=30,
            num_total_passages=100,
        ):
        # Value is fixed to simplify signature construction in
            presented snippet
        self.num_search_queries_per_hop = 3

        self.num_hops = num_hops
        self.k_per_search_query = k_per_search_query
        self.k_per_search_query_last_hop = k_per_search_query_last_hop
        self.num_total_passages = num_total_passages

        self.rm = dspy.ColBERTv2()
        self.generate_query = dspy.ChainOfThought(GenerateThreeQueries)
        self.append_notes = dspy.ChainOfThought(AppendNotes)

    def forward(self, claim: str) -> list[str]:
        key_facts = []
        committed_docs = []

        for hop_ind in range(self.num_hops):
            is_last_hop = hop_ind == self.num_hops - 1
            is_first_hop = hop_ind == 0
            hop_k = self.k_per_search_query_last_hop if is_last_hop else self.k_per_search_query
            num_docs_to_keep = (self.num_total_passages - len(committed_docs)) if is_last_hop else self.k_per_search_query

            if is_first_hop:
                search_queries = [claim]
            else:
                pred = self.generate_query(claim=claim, key_facts=key_facts)
                search_queries = [pred.search_query1, pred.search_query2, pred.search_query3]
            search_queries = deduplicate(search_queries)

```

```
            search_results = [r for q in search_queries for r in
                search_raw(q, k=hop_k, rm=self.rm)]
            search_results = sorted(search_results, key=lambda r:
                r["score"], reverse=True)

            unique_docs = []
            for result in search_results:
                if result["long_text"] not in unique_docs:
                    unique_docs.append(result["long_text"])
            unique_docs = unique_docs[:num_docs_to_keep]
            committed_docs.extend(unique_docs)

            if not is_last_hop:
                pred = self.append_notes(claim=claim, key_facts=
                    key_facts, new_search_results=unique_docs)
                key_facts.append(pred.new_key_facts)

        return dspy.Prediction(key_facts=key_facts, retrieved_docs
            =committed_docs)
```

Python Snippet 3: DSPy program for HoVer.

C Asset information

The license information for the models and datasets we used are shared below. All models and datasets are access via HuggingFace.

`qwen3-8b` is shared with the `Apache License 2.0`, accessed via the HuggingFace model identifier `Qwen/Qwen3-8B`

`llama3.1-8b-instruct` is shared with the `Meta Llama 3 Community License`, accessed via the HuggingFace model identifier `meta-llama/Meta-Llama-3.1-8B-Instruct`

`Banking77` is shared with `CC BY 4.0` license

`HoVer` is shared with `CC BY 4.0` license

`PAPILLON` is shared with the `MIT License` license

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract claims (i) a multi-module generalization of GRPO that groups LM calls by module and handles variable-length/interrupted trajectories, and (ii) that composing mmGRPO with BetterTogether improves accuracy over a post-trained LM and over prompt optimization alone across classification, many-hop search, and privacy-preserving delegation; it also (iii) states the code release. (i) is introduced and formalized in Section 3; the composition with BetterTogether is described in Section 4; and the empirical gains are supported in Table 1 with average improvements of 11% vs. the post-trained LM and 5% vs. prompt optimization alone.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.

- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Explained in Section 8.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: Paper does not include theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Our approach is described fully in Section 3, Section 4, and Appendix A, and experimental details are shared in Section 5. Our results can be reproduced by following these recipes, but we also release our optimizer in an easy to use library.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our optimizer is publicly released in the DSPy library at `dspy.ai`. We also provide a repository with more instructions on how to run the experiments reported in our paper.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Details related to datasets and experiments are shared in Section 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: All experiments are repeated 3 times for reproducibility, but no error bars were reported.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [No]

Justification: Paper provides details about the average run time and GPUs used, but does not provide experiment-level stats.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: Paper adheres to NeurIPS code of ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: The paper studies how to make Natural Language Processing systems more reliable for a set of specific tasks.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Not relevant for the paper.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: See Appendix C.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: mmGRPO optimizer is documented in the DSPy library.

    Guidelines:

    - The answer NA means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Paper does not crowdsource experiments.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Paper does not include research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: For one of the datasets in the paper, LLMs were used as judges, details of which are shared in Section 5 and Appendix B.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.

- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.