AutoDSPy: Automating Modular Prompt Design with Reinforcement Learning for Small and Large Language Models

Nafew Azim[1] **Abrar Ur Alam**[1] **Hasan Bin Omar**[1]
Abdullah Mohammad Muntasir Adnan Jami[1] Jawad Ibn Ahad[1]
Muhammad Rafsan Kabir[1] Md. Ismail Hossain[1] Fuad Rahman[2]
Mohammad Ruhul Amin[3] Shafin Rahman[1] Nabeel Mohammed[1]
[1]North South University [2]Apurba Technologies [3]Fordham University
{nafew.azim, abrar.alam, hasan.omar, abdullah.jami, jawad.ibn, muhammad.kabir,
ismail.hossain2018, shafin.rahman, nabeel.mohammed}@northsouth.edu,
fuad@apurbatech.com, mamin17@fordham.edu

Abstract

Large Language Models (LLMs) excel at complex reasoning tasks, yet their performance hinges on the quality of their prompts and pipeline structures. Manual prompt design, as used in frameworks like DSPy, poses significant limitations: it is time-intensive, demands substantial expertise, and lacks scalability, restricting the widespread use of LLMs across diverse applications. To overcome these challenges, we introduce AutoDSPy, the first framework to fully automate DSPy pipeline construction using reinforcement learning (RL). AutoDSPy leverages an RL-tuned policy network to dynamically select optimal reasoning modules—such as Chain-of-Thought for logical tasks or ReAct for tool integration—along with input-output signatures and execution strategies, entirely eliminating the need for manual configuration. Experimental results on the GSM8K and HotPotQA benchmarks demonstrate that AutoDSPy outperforms traditional DSPy baselines, achieving accuracy gains of up to 4.3% while reducing inference time, even with smaller models like GPT-2 (127M). By integrating RL-based automation, AutoDSPy enhances both efficiency and accessibility, simplifying the development of structured, high-performing LLM solutions and enabling scalability across a wide range of tasks. The code is available at https://github.com/nafew-azim/AUTODSPy (DOI: 10.5281/zenodo.17276875).

Keywords: Large Language Models, Prompt Optimization, Reinforcement Learning, AutoDSPy, Chain-of-Thought

1 Introduction

Large Language Models (LLMs) have showcased extraordinary abilities in generating human-like text and tackling complex reasoning tasks. Yet, their effectiveness hinges heavily on how prompts are crafted and pipelines are configured. Manual prompt engineering (Marvin et al., 2023) is a labor-intensive process, often requiring extensive trial-and-error to refine prompts, deep expertise to navigate design choices, and significant time investment—sometimes yielding suboptimal results. These challenges intensify in scenarios demanding quick deployment or intricate reasoning, such as mathematical problem-solving or multi-hop question answering, where poorly designed pipelines can degrade accuracy and inflate computational costs.

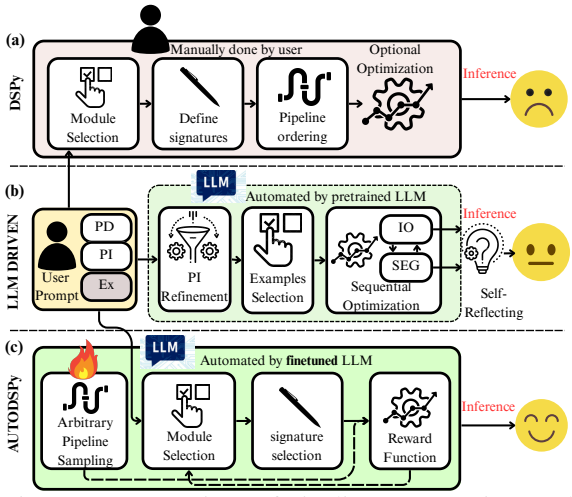

Figure 1: Comparison of pipeline construction methods: (a) Traditional DSPy (Khattab et al., 2024) requires manual, static configuration of modules and signatures. (b) LLM-driven refinement (Agarwal et al., 2024) iteratively improves prompts using LLM reasoning. (c) Our method, AutoDSPy, uses reinforcement learning to train a policy network that builds optimized pipelines based on execution feedback.

To address these challenges and make LLMs more accessible to a broader user base, several frameworks have emerged to support structured prompting. These frameworks offer modular components, enabling users to build workflows that leverage advanced strategies like Chain-of-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 2881–2896
November 4-9, 2025 ©2025 Association for Computational Linguistics

Thought (CoT) reasoning (Wei et al., 2022) or tool-augmented interaction. Despite their advantages, they still demand significant manual effort: users must specify input-output signatures, select suitable modules, and fine-tune parameters. These tasks are tedious, prone to errors, and difficult to scale across varied tasks or domains.

DSPy (Khattab et al., 2024) was introduced to alleviate the challenges of manual prompt engineering by providing a structured and modular framework for building large language model (LLM) pipelines. It incorporates predefined modules that implement advanced prompting techniques, such as Chain-of-Thought (CoT) (Wei et al., 2022) and ReAct (Yao et al., 2023), and facilitates signature-based definitions of input-output schemas. This design empowers practitioners, particularly those lacking extensive expertise in prompt engineering, to construct robust reasoning workflows with minimal effort. Nevertheless, DSPy retains a dependency on manual decision-making for essential aspects, including the selection of modules, the specification of signatures, and the choice of optimization strategies. Such manual requirements constrain DSPy's scalability, especially in dynamic or heterogeneous task environments where configuration complexity can increase significantly.

Several existing methods have sought to automate aspects of prompt engineering, yet each falls short in key areas critical for scalable and flexible LLM pipeline construction. Fig. 1 illustrates these trade-offs by comparing traditional prompt refinement methods with our automated approach. For instance, *(a)* PromptWizard (Agarwal et al., 2024) optimizes instruction templates and in-context examples for single LLM invocations using LLM-based evaluation (Li et al., 2024a). However, it does not extend to constructing multi-stage pipelines and incurs high computational costs due to frequent model calls. Similarly, *(b)* EvoPrompt (Guo et al., 2023) efficiently synthesizes improved prompts by combining high-performing candidates but lacks the programmability and adaptability inherent in modular pipeline frameworks.*(c)* RLPrompt (Deng et al., 2022), while leveraging reinforcement learning (Kaelbling et al., 1996) to enhance prompt utilization, does not offer the modular control and extensibility that pipeline-based systems provide. These limitations highlight the need for a more comprehensive solution, addressed by our proposed AutoDSPy framework.

To address the inherent limitations of manual pipeline design in DSPy, we present AutoDSPy—a principled framework extension that fully automates the creation of modular, task-adaptive large language model (LLM) pipelines. At its core, AutoDSPy leverages a Policy Network (PN), powered by a reinforcement learning (RL)-tuned LLM, to manage the sequential decision-making process for pipeline synthesis. This PN independently chooses the most effective reasoning modules—such as Chain-of-Thought for logical tasks—assigns optimal input-output signature mappings, and crafts tailored task execution strategies, eliminating the need for manual adjustments previously required in DSPy. The result is a flexible, self-configuring pipeline that adapts effortlessly to diverse tasks with minimal human input. Extensive testing across a range of reasoning-intensive datasets reveals that AutoDSPy not only matches DSPy's structured reasoning abilities but consistently exceeds its performance in accuracy and efficiency. By blending automation with modularity, AutoDSPy sets a new standard for scalability, efficiency, and usability in LLM-based systems, unlocking powerful solutions for complex, multi-step problem-solving.

Contributions: The significant contributions are summarized as follows -

- **AutoDSPy Framework:** The first fully automated extension of DSPy, using RL to eliminate manual pipeline configuration.
- **Policy Network Design:** An RL-tuned LLM that dynamically selects modules, signatures, and strategies for task-adaptive pipeline synthesis.
- **Empirical Superiority:** Proven gains in accuracy and efficiency on benchmarks like GSM8K and HotPotQA, surpassing manual DSPy while preserving flexibility.

2 Related Work

Reinforcement Learning: Prompt optimization has made significant progress through reinforcement learning (RL), evolutionary algorithms, and strategic prompting techniques. RL-based strategies like REINFORCE (Williams, 1992), PPO (Schulman et al., 2017), and GRPO (Shao et al., 2024) have demonstrated stable optimization strategies. Studies utilized RL for prompt fine-tuning in the guise of AutoPrompt (Shin et al., 2020), Prewrite (Kong et al., 2024), and RLPrompt (Deng

et al., 2022), while AutoFlow (Li et al., 2024b) and GRL-Prompt (Liu et al., 2024) employ RL for workflow automation and knowledge graph integration. Nonetheless, such models typically are not programmable and need retraining for strategy customization. AutoDSPy overcomes this by training an RL-based policy network (PN) that creates DSPy pipelines (Khattab et al., 2024), with users having complete control to customize afterwards.

Evolutionary Algorithms: Another effective option is evolutionary algorithms. EvoPrompt (Guo et al., 2023), PhaseEvo (Cui et al., 2024), and PromptWizard (Agarwal et al., 2024) learn and develop prompts iteratively, based on performance, which tend to be better than human methods. Their use of repeated LLM calls for criticism, variation, and synthesis, however, result in large computational costs. AutoDSPy solves this by learning an LLM to create pipelines with comparable strategies without repeated LLM calls.

Prompting Strategies: Studies by ProTeGi (Pryzant et al., 2023), BetterTogether (Soylu et al., 2024), Adversarial In-Context Learning (Long et al., 2024), and SoftPromptComp (Wang et al., 2024), explore diverse prompting strategies from feedback-based prompting to adversarial games and context compression. However, they suffer from the inability to control and interpret or demand plenty of manual labor and domain expertise. AutoDSPy bridges this gap by combining RL with DSPy's framework, making the pipeline development automatic but still user-programmable and supporting customization at an efficient cost and thus being worthy of usage.

3 Methodology

3.1 Problem Formulation

The current DSPy framework depends on manual configuration of reasoning modules (e.g., Chain-of-Thought, Predict), input-output signatures (e.g., `question → answer`), and teleprompters (example-based optimizers). This manual design process introduces inefficiencies, requires expert intervention, and often results in suboptimal performance.

Formally, the objective is to learn how to construct an optimal pipeline using a set of reasoning modules $\mathcal{M} = \{M_0, M_1, \ldots, M_n\}$, where each M_i represents a distinct reasoning strategy; a set of signatures $\mathcal{S} = \{s^0, s^1, \ldots\}$, where each s_j defines an input-output mapping; and a set of teleprompters $\mathcal{T} = \{t_0, t_1, \ldots, t_l\}$, where each t_p governs how examples are retrieved or generated. A pipeline P of length L is defined as a sequence of module-signature pairs optimized under a selected teleprompter:

$$P = \big((M_0, s_0), (M_1, s_1), \ldots, (M_{L-1}, s_{L-1}); t\big),$$

where each $M_k \in \mathcal{M}$, $s_k \in \mathcal{S}$, and $t \in \mathcal{T}$. Given a dataset $(x, y) \sim \mathcal{D}$ of input–ground truth pairs, the pipeline executes as:

$$\begin{aligned} o_0 &= M_0^{s_0}(x), \\ o_1 &= M_1^{s_1}(o_0), \\ &\vdots \\ o_{L-1} &= M_{L-1}^{s_{L-1}}(o_{L-2}), \end{aligned}$$

with final output o_{L-1}. The goal is to ensure that $o_{L-1} \approx y$.

Solution Strategy: To automate this process, we introduce a PN π_θ, parameterized by a finetuned LLM, which maps an input x to a pipeline, $P = \pi_\theta(x) = \big((M_0, s_0), (M_1, s_1), \ldots, (M_{L-1}, s_{L-1}); t\big)$. We aim to learn θ such that for a random pair $(x, y) \sim \mathcal{D}$, the pipeline's final output o_{L-1} matches the target y with high probability. In a RL setting, this corresponds to defining a reward, $R(o_{L-1}, y) = \mathbf{1}\{o_{L-1} = y\}$, and maximizing the expected return, $J(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \; \mathbb{E}_{P\sim\pi_\theta(\cdot|x)}\big[R(o_{L-1}, y)\big]$. The objective of this work is to learn such a PN π_θ that constructs task-specific pipelines end-to-end, automatically selecting and composing reasoning modules, signatures, and teleprompters to maximize performance without manual configuration.

3.2 Revisiting DSPy

DSPy (Khattab et al., 2024) is a declarative programming framework designed to transform language model (LM) invocations into self-improving pipelines. It abstracts LM workflows using three key components: *signatures*, *modules*, and *teleprompters*, each of which contributes to defining, executing, and optimizing text transformation tasks. Formally, a DSPy program consists of a sequence of module calls over natural language signatures. Each **signature** $s \in \mathcal{S}$ defines a transformation schema as a typed input-output mapping, such as `question → answer`. This abstraction specifies what task is to be performed, rather than how the underlying LM is to be

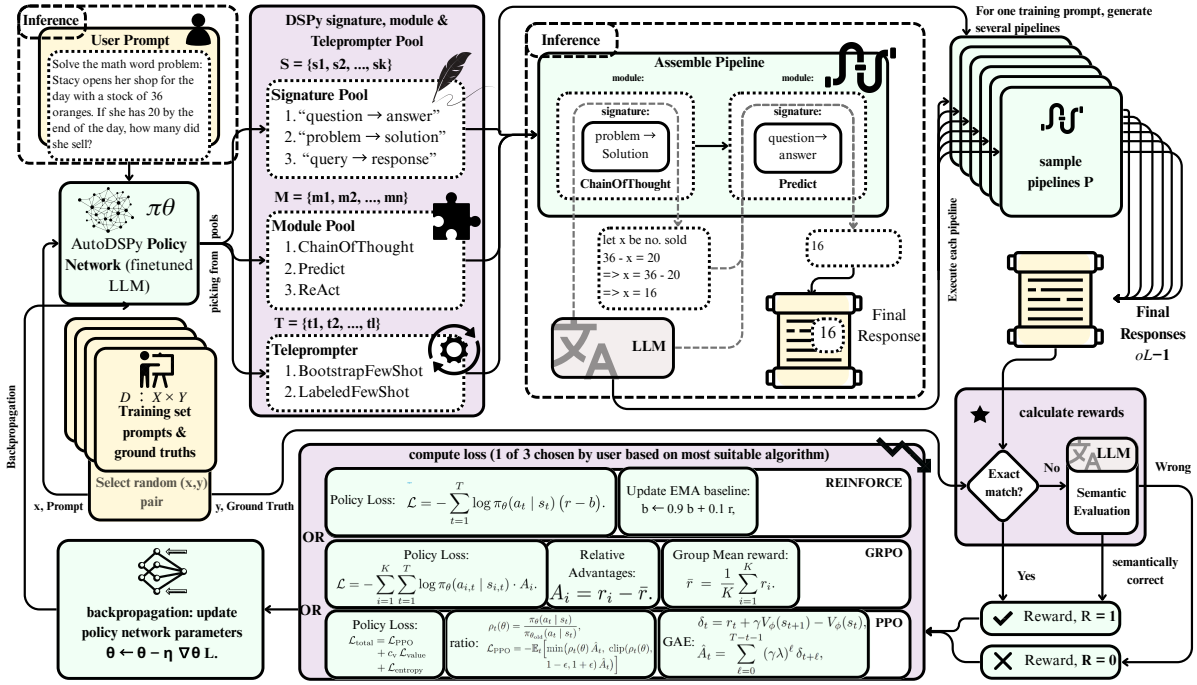

Figure 2: AutoDSPy detailed workflow: inference-generation portion in dashed-line region, and network training portion outside. Training randomly selects an (x, y) prompt-groundtruth pair from training dataset $D : X \times Y$, using prompt x the PN $\pi_\theta(x)$ generates several sample pipelines P from DSPy's pool of modules, signatures and teleprompters M, S and T respectively, then the pipeline reward is computed on their final output o_{L-1} using $R(o_{L-1}, y) = \mathbf{1}\{o_{L-1} = y\}$, the reward R is further used to calculate loss in one of three ways (REINFORCE, GRPO or PPO - whichever the user deems most fit), finally the loss $\mathcal{L}$ is used to backpropagate through the PN π_θ updating its parameters $\theta \leftarrow \theta - \eta \nabla \theta L$ and this cycle repeats for N episodes. Inference simply takes the user's prompt to generate one pipeline from the PN, and runs an LLM through it.

prompted. A **module** $M \in \mathcal{M}$ implements the logic to fulfill a signature by invoking an LM through prompting or finetuning. Modules include declarative instantiations of prompting strategies, such as `Predict`, `ChainOfThought` (Wei et al., 2022), `ReAct` (Yao et al., 2023), and `ProgramOfThought` (Chen et al., 2023), among others. Each module is parameterized by an LM and a set of demonstrations, and is executable as a function $M^s(x)$, where x satisfies the input type of s. Given a sequence of L such pairs $((M_0, s_0), (M_1, s_1), \ldots, (M_{L-1}, s_{L-1}))$, a pipeline P can be viewed as: $P = \big((M_0, s_0), (M_1, s_1), \ldots, (M_{L-1}, s_{L-1}); t\big)$, where $t \in \mathcal{T}$ is a **teleprompter** that defines the optimization strategy used to improve the modules' behavior. teleprompters simulate and collect demonstration traces to guide the learning or prompt construction process, often with a reward metric for end-task quality. The execution of such a pipeline over input x proceeds recursively with the final output o_{L-1} evaluated against a ground-truth label y using a task-specific metric.

In essence, DSPy shifts the focus from hand-crafted prompt strings to high-level, declarative specifications of transformation goals. Its ability to adaptively optimize module behavior through demonstration-based or fine-tuning-based strategies enables the construction of robust and scalable LM pipelines across a wide range of reasoning tasks. However, DSPy still requires manual selection of modules $M_k \in \mathcal{M}$, signatures $s_k \in \mathcal{S}$, and teleprompters $t \in \mathcal{T}$ to define a pipeline $P = ((M_0, s_0), \ldots, (M_{L-1}, s_{L-1}); t)$. The core problem we aim to solve is the automation of this pipeline synthesis by learning a policy $\pi_\theta(x)$ that maps an input x to an optimal pipeline P such that the expected output $\mathbb{E}_{P \sim \pi_\theta(x)}[o_{L-1}]$ maximizes a task-specific reward: $\theta^* = \arg\max_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathbb{E}_{P\sim\pi_\theta(x)} \left[R(o_{L-1}, y)\right]\right]$ ..

3.3 Implication of the AutoDSPy Framework

AutoDSPy extends the existing DSPy framework (Khattab et al., 2024) by presenting an automation of pipeline synthesis with a specific focus on tasks for large language models (LLMs) through the use of a policy network (PN) π_θ. The policy network is an LLM fine-tuned to generate

pipelines as $P = ((M_1, s_1), \ldots, (M_L, s_L); t)$, where $M_\ell \in \mathcal{M}$, $s_\ell \in \mathcal{S}$, and $t \in \mathcal{T} \cup \{\varnothing\}$ are reasoning modules, input-output signatures, and an optional teleprompter, respectively. Pipeline generation is cast as a sequential decision process in reinforcement learning (RL), where actions select components. If a teleprompter is chosen, a synthetic training set is constructed; otherwise, zero-shot execution occurs. The PN is learned to maximize expected reward: $\theta^* = \arg\max_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \mathbb{E}_{P\sim\pi_\theta(x)}[R(o_{L-1}, y)]$, where $R(o_{L-1}, y)$ assesses pipeline output against ground truth. This enables automated, high-quality pipeline synthesis (see Fig. 2). Notably, We used binary rewards (R=1 for correct, R=0 otherwise) with semantic fallback to simplify training while capturing partial correctness. This design balances efficiency and robustness, as semantic fallback via LLM assessment ensures nuanced feedback for non-exact matches, avoiding complex reward engineering.

Policy Network Building. The PN π_θ, which is constructed over a fine-tuned LLM, translates input prompts into pipelines. Seven transformer-based LLMs, described in the Implementation section, were tested for encoding state-action trajectories and producing valid decisions in limited action spaces and synthesizing pipelines with optimal reasoning tasks.

RL Algorithms. Three RL methods train the PN: (1) REINFORCE (Zhang et al., 2021) optimizes π_θ through stochastic gradient ascent with an EMA baseline; (2) PPO (Schulman et al., 2017) stabilizes using clipped objectives and entropy regularization; (3) GRPO (Shao et al., 2024) reduces variance by normalizing rewards within prompt-specific groups. All three methods construct pipelines by sampling actions, executes them, and updating π_θ on reward.

Prompt Refinement. AutoDSPy indirectly refines prompts by the selection and composition of modules and signatures that structure LLM reasoning, refining prompt quality by component selection rather than direct string edits.

Evaluation and Inference. The performance of the pipeline is measured in terms of accuracy and latency relative to benchmark tasks. During inference, the PN generates a pipeline, executed with an LLM to produce final output, while providing DSPy compatibility and dynamic prompt adaptation.

Table 1: Performance comparison of the proposed AutoDSPy framework against DSPy methods such as the original modules (Khattab et al., 2024), MIPROv2 teleprompter (Opsahl-Ong et al., 2024) and BetterTogether teleprompter (Soylu et al., 2024). AutoDSPy uses three RL strategies — REINFORCE (♢), PPO (♡), and GRPO (♣)—on the GSM8K and HotPotQA benchmarks. No teleprompters were used for AutoDSPy. **Bold Red** values indicate the best performance; Blue values indicate the second best. Generalizability across diverse benchmarks is evaluated in Appendix A.2.

Method	RL	LLaMA-3.1 - 8B		Qwen-2.5 - 14B	
		GSM8K Acc / Time (%) / (s)	HotPotQA Acc / Time (%) / (s)	GSM8K Acc / Time (%) / (s)	HotPotQA Acc / Time (%) / (s)
DSPy Methods					
Predict	-	70.6 / 6.4	70.5 / 7.4	71.3 / 6.2	71.1 / 7.1
CoT	-	79.9 / 11.1	72.3 / 8.5	80.1 / 10.8	73.5 / 8.1
MIPROv2	-	76.1 / 13.9	29.6 / 9.2	76.9 / 13.2	30.6 / 9.1
BetterTogether*		79.8 / 9.2	31.6 / 4.6	80.4 / 9.0	31.8 / 4.3
AutoDSPy RL-Finetuned LMs (Ours)					
GPT-2-127M	♢	76.8 / **5.1**	74.2 / 6.1	77.2 / **4.9**	74.8 / 5.9
	♡	57.7 / 7.9	75.9 / 8.1	58.1 / 7.5	76.5 / 7.9
	♣	**82.4** / 8.1	73.9 / **4.2**	**83.0** / 8.0	74.3 / **4.0**
LLaMA-3.2-1B	♢	75.5 / 7.4	73.0 / 6.4	76.2 / 7.2	73.8 / 6.2
	♡	79.7 / 8.1	73.9 / 7.4	80.2 / 7.9	74.1 / 7.2
	♣	79.5 / 8.9	74.0 / 10.5	80.4 / 8.7	74.9 / 10.0
Gemma-3-1B	♢	77.0 / 6.0	74.0 / 6.5	77.5 / 5.8	74.5 / 6.2
	♡	78.5 / 7.8	74.5 / 7.5	79.0 / 7.6	75.0 / 7.3
	♣	80.2 / 8.5	75.0 / 5.0	80.7 / 8.3	75.5 / 4.8
DeepSeek-R1-1.5B	♢	80.6 / 7.7	73.1 / 8.0	81.1 / 7.5	74.0 / 7.8
	♡	79.7 / 9.8	73.0 / 8.5	80.0 / 9.5	73.7 / 8.1
	♣	69.2 / 10.5	**76.6** / 13.2	70.1 / 10.1	**77.2** / 13.0
Qwen-2.5-1.5B	♢	78.0 / 7.0	75.5 / 7.0	78.5 / 6.8	76.0 / 6.8
	♡	80.0 / 8.2	74.1 / 8.5	80.5 / 8.0	76.5 / 8.3
	♣	70.3 / 9.0	76.0 / 6.0	71.4 / 8.8	76.7 / 5.8
Mistral-v0.1-7B	♢	77.5 / 7.5	75.0 / 7.2	78.0 / 7.3	75.5 / 7.0
	♡	79.0 / 8.5	75.5 / 8.0	79.5 / 8.3	76.0 / 7.8
	♣	79.9 / 9.5	75.5 / 6.5	82.0 / 9.3	77.0 / 6.3
Phi-4-14B	♢	76.5 / 6.5	73.5 / 6.8	77.0 / 6.3	74.0 / 6.6
	♡	78.0 / 8.0	74.0 / 7.8	78.5 / 7.8	74.5 / 7.6
	♣	80.5 / 8.8	74.5 / 5.5	81.0 / 8.6	75.0 / 5.3

* teleprompters used in conjunction with DSPy-CoT pipeline

4 Experiments

4.1 Setup

Datasets: We evaluated AutoDSPy on two benchmark datasets: GSM8K (Cobbe et al., 2021) for mathematical reasoning and HotPotQA (Yang et al., 2018) for multi-hop question answering. These datasets are well-suited for assessing both arithmetic precision and complex language understanding. Following the experimental setup in (Khattab et al., 2024), we fine-tuned models on 200 training samples and evaluated them on 1,300 (GSM8K) and 1,000 (HotPotQA) test samples. See Appendix A.3 for implementation details.

Evaluation Metrics: To assess the performance of

our proposed AutoDSPy pipeline, we measured accuracy and inference time. Accuracy was evaluated using exact match against ground truth, consistent with the evaluation methodology in DSPy (Khattab et al., 2024). To account for cases where exact match may not capture semantic equivalence, we incorporated an LLM-based evaluation as a fallback. Inference time, defined as the average time required to process a single test instance, was included to demonstrate that AutoDSPy maintains practical inference speeds. See Appendix A.4 for hyperparameter settings and configuration details.

4.2 Main Results

This study evaluates the AutoDSPy framework, which leverages three reinforcement learning strategies—Reinforce (♢), PPO (♡), and GRPO (♣)—across diverse policy language models on GSM8K and HotPotQA benchmark datasets for mathematical reasoning and multi-hop question answering, respectively. AutoDSPy demonstrates superior performance compared to baseline DSPy methods, with GRPO (♣) achieving 82.4% accuracy on GSM8K (vs. DSPy-CoT's 79.9%) while reducing inference time from 11.1s to 8.1s, and 77.2% accuracy on HotPotQA (vs. DSPy-CoT's 73.5%). The RL strategies exhibit distinct optimization profiles: GRPO (♣) consistently delivers the highest accuracy (83.0% on GSM8K) by optimizing multiple pipeline candidates but incurs longer inference times with larger models; Reinforce (♢) prioritizes computational efficiency with the fastest inference times (5.1s on GSM8K) while maintaining competitive accuracy; and PPO (♡) balances accuracy and efficiency through clipped surrogate objectives, offering robust performance across tasks with second-tier accuracy on both datasets. Policy model selection significantly influences performance, with smaller models excelling on GSM8K's step-by-step reasoning due to efficiency, while larger models dominate HotPotQA's multi-hop challenges through enhanced context-handling capabilities.The choice of inference model further impacts results, with larger models generally improving accuracy by capturing nuanced patterns, as evidenced by consistent performance gains across configurations. Baseline methods reveal notable limitations: DSPy-Predict (Khattab et al., 2024) achieves moderate accuracy (70.6% on GSM8K) but lacks reasoning depth; DSPy-CoT (Khattab et al., 2024) improves accuracy (79.9% on GSM8K) through step-by-step reasoning but incurs longer inference times; DSPy-CoT + MIPROv2 (Opsahl-Ong et al., 2024) struggles with multi-hop reasoning on HotPotQA (29.6%); and DSPy-CoT + BetterTogether (Soylu et al., 2024) performs competitively on GSM8K (80.4%) but falters on HotPotQA (31.8%). These findings affirm AutoDSPy's efficacy in integrating RL strategies with carefully selected policy models to achieve superior accuracy and efficiency, consistently outperforming baselines and establishing a new benchmark for programmatic language model optimization in mathematical reasoning and multi-hop question answering tasks. Among the RL-algorithms, GRPO demonstrates a favorable balance between performance gains and training complexity. Although RL introduces additional training overhead, it enables the discovery of adaptive and efficient pipeline structures that generalize well across different tasks. These results highlight RL's capacity in optimizing modular language model systems, paving the way for more efficient and robust reasoning frameworks in future research.See Appendices A.1, A.6, and A.7 for ablation study, computational analysis, and statistical analysis, respectively.

5 Conclusion

In this study, we introduced an RL-driven extension of the DSPy framework that automates the design and optimization of reasoning pipelines for complex tasks such as mathematical problem solving and multi-hop question answering. AutoDSPy moves beyond static prompting methods to achieve improvements in both performance and efficiency. We demonstrate that RL can discover effective reasoning structures without manual intervention which offers greater flexibility and scalability across diverse tasks. AutoDSPy thus represents a step forward in developing dynamic, adaptable pipelines for NLP. Future research on AutoDSPy will aim to expand the module-signature library and optimize training costs through techniques such as early stopping, curriculum learning, or lightweight reinforcement learning variants. The scope of the framework will be extended to domains such as code synthesis, scientific reasoning, and multimodal tasks. Furthermore, rigorous benchmarking will be performed against established baselines and open domain prompting methods to comprehensively evaluate the strengths and limitations of AutoDSPy.

6 Ethics Statement

This work proposes AutoDSPy, a framework that automates the construction of language model pipelines using RL. While our methods aim to improve reasoning quality and reduce manual prompt engineering, we acknowledge the ethical implications of automated decision-making in language models. All experiments are conducted on publicly available datasets and adhere to open-source model licenses. We do not use or fine-tune on sensitive, private, or personally identifiable information. Nonetheless, as AutoDSPy relies on pretrained LLMs, it inherits the biases present in the underlying models. We caution against the deployment of AutoDSPy in safety-critical or socially sensitive domains without appropriate safeguards, interpretability tools, and human oversight. Furthermore, although AutoDSPy improves efficiency, care must be taken to prevent misuse in automating misinformation, harmful content generation, or biased decision systems. Future work should continue to explore mechanisms for responsible alignment, auditability, and fairness in automated LLM pipeline design.

7 Limitations

Despite these positive findings, AutoDSPy still has a variety of important restrictions that constrain its more universal applicability and usage. Firstly, the implementation depends on a relatively limited and finite number of DSPy modules and natural language signatures, restricting the versatility of the system. In addition, current teleprompters are neither flexible nor robust in dynamically guiding prompt construction across a wide range of tasks and thinking styles. As a result, the diversity of compositional strategies that AutoDSPy is able to browse remains limited. Second, the RL algorithms, while effective in optimizing module selection and prompt crafting, are computationally intensive. This complicates scaling the strategy to bigger models, more advanced pipelines, or use in low-resource environments such as edge devices or sparse cloud infrastructure. Third, while our expanded evaluation across six benchmarks (GSM8K, HotPotQA, MGSM, MBPP, LegalBench, PubMedQA) demonstrates improved generalizability, questions remain about performance on non-textual modalities, extremely low-resource languages, or highly specialized domains. These are important challenges to overcome to enhance the scalability, robustness, and flexibility of AutoDSPy in real-world scenarios.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, and 8 others. 2024. Phi-4 technical report. *Preprint*, arXiv:2412.08905.

Eshaan Agarwal, Joykirat Singh, Vivek Dani, Raghav Magazine, Tanuja Ganu, and Akshay Nambi. 2024. Promptwizard: Task-aware prompt optimization framework. *arXiv preprint arXiv:2405.18369*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Wendi Cui, Jiaxin Zhang, Zhuohang Li, Hao Sun, Damien Lopez, Kamalika Das, Bradley Malin, and Sricharan Kumar. 2024. Phaseevo: Towards unified in-context prompt optimization for large language models. *arXiv preprint arXiv:2402.11347*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3379, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Neel Guha, Julian Nyarko, Daniel E. Ho, Christopher Ré, Adam Chilton, Aditya Narayana, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel N. Rockmore, Diego Zambrano, Dmitry Talisman, Enam Hoque, Faiz Surani, Frank Fagan, Galit

Sarfaty, Gregory M. Dickinson, Haggai Porat, Jason Hegland, and 21 others. 2023. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Preprint*, arXiv:2308.11462.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Qingyan Guo, Rui Wang, Guo Junliang, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *arXiv preprint arXiv:2309.08532*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146*.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, and 1 others. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

Weize Kong, Spurthi Hombaiah, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024. Prewrite: Prompt rewriting with reinforcement learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 594–601.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024a. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024b. Autoflow: Automated workflow generation for large language model agents. *CoRR*.

Yuze Liu, Tingjie Liu, Tiehua Zhang, Youhua Xia, Jinze Wang, Zhishu Shen, Jiong Jin, and Fei Richard Yu. 2024. Grl-prompt: Towards knowledge graph based prompt optimization via reinforcement learning. *arXiv preprint arXiv:2411.14479*.

Do Long, Yiran Zhao, Hannah Brown, Yuxi Xie, James Zhao, Nancy Chen, Kenji Kawaguchi, Michael Shieh, and Junxian He. 2024. Prompt optimization via adversarial in-context learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7308–7327.

Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. 2023. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*, pages 387–402. Springer.

Krista Opsahl-Ong, Michael Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, and 1 others. 2022. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235.

Dilara Soylu, Christopher Potts, and Omar Khattab. 2024. Fine-tuning and prompt optimization: Two

great steps that work better together. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10696–10710.

Gemma Team. 2024. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*. Correspondence to gemma-3-report@google.com.

Cangqing Wang, Yutian Yang, Ruisi Li, Dan Sun, Ruicong Cai, Yuzhu Zhang, and Chengqian Fu. 2024. Adapting llms for efficient context processing through soft prompt compression. In *Proceedings of the International Conference on Modeling, Natural Language Processing and Machine Learning*, pages 91–97.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. 2021. Sample efficient reinforcement learning with reinforce. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10887–10895.

A Appendix

A.1 Ablation Study

To evaluate the impact of key components in our proposed framework, we conducted ablation experiments on four distinct aspects: (a) the effect of training episodes (b) the effect of training samples (c) the effect of using a teleprompter and (d) the effect of GRPO Hyperparameter k.

Effectiveness of Number of Training Episodes: In our ablation study, we evaluated the effect of varying the number of reinforcement learning (RL) training episodes (200 vs. 300), with fixed train-

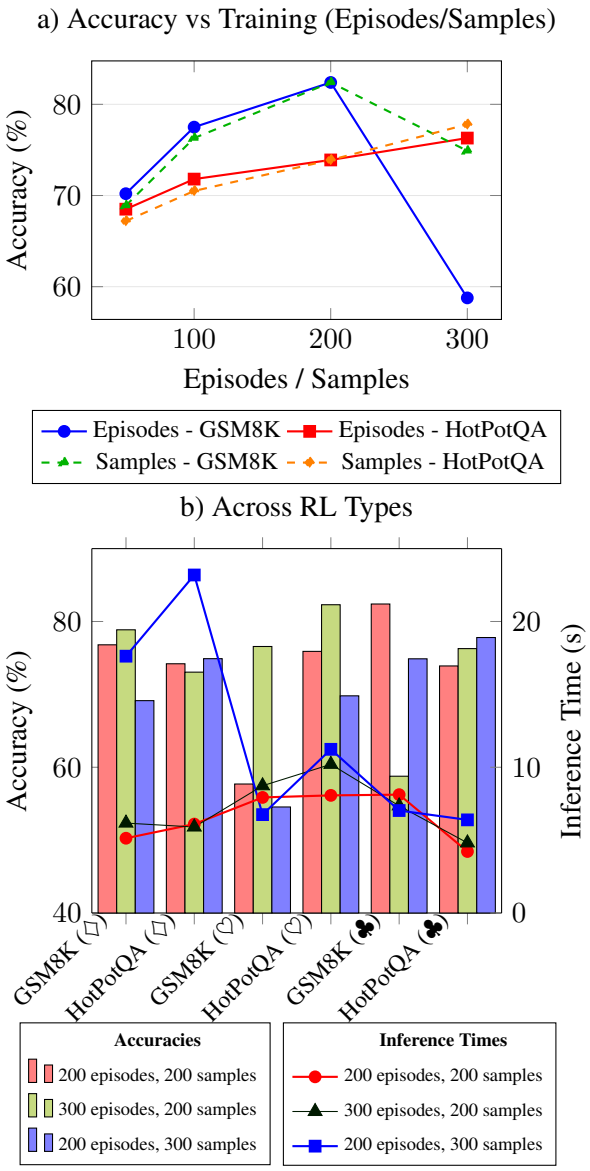

Figure 3: a) Learning curves showing accuracy vs training (episodes/samples) on GPT-2-127M with GRPO, inference with LLaMA-3.1-8B. With episodes or samples held constant at 200 when varying the other, accuracy increases steadily to 200 episodes/samples and mostly plateaus at 300. However, 300 training episodes causes a significant accuracy drop on GSM8K (from 82.4% to 58.8%), indicating overfitting and the need for precise episode tuning. b) Impact of varying RL training episodes and samples on GPT-2 performance across RL algorithms: REINFORCE (♢), PPO (♡), and GRPO (♣), using LLaMA-3.1-8B for inference. Measured by accuracy (%) and inference time (s) on GSM8K and HotPotQA datasets. PPO (♡) excels with more episodes, showing accuracy improvements from 57.7% to 76.6% on GSM8K. Larger samples often reduce accuracy across algorithms, with inconsistent inference times.

ing samples (200), across Reinforce (♢), PPO (♡), and GRPO (♣) algorithms on the GSM8K and HotPotQA datasets. As shown in Figure 3, increasing episodes from 200 to 300 generally enhanced accuracy for PPO (♡), particularly on GSM8K, where accuracy improved from 57.7% to 76.6%, due to improved policy optimization. However, GRPO (♣) and Reinforce (♢) exhibited mixed results, with GRPO (♣) showing accuracy declines on GSM8K from 82.4% to 58.8% but gains on HotPotQA from 73.9% to 76.3%, and Reinforce (♢) yielding modest accuracy improvements on GSM8K from 76.8% to 78.9% or slight declines on HotPotQA from 74.2% to 73.1%. Inference times showed marginal changes, with minor increases or decreases depending on the algorithm and dataset; for example, PPO (♡) on GSM8K increased from 7.9s to 8.7s, while GRPO (♣) on HotPotQA changed from 4.2s to 4.8s (see Figure 3). These findings suggest that while additional episodes benefit certain RL strategies like PPO (♡), effects vary, underscoring the need for careful episode tuning to balance accuracy and computational cost.

Effectiveness of Training Sample Sizes: We also examined the effect of varying training sample sizes (200 vs. 300), with fixed training episodes (200), across Reinforce (♢), PPO (♡), and GRPO (♣) algorithms on the GSM8K and HotPotQA datasets. As illustrated in Figure 3, increasing samples from 200 to 300 typically reduced accuracy for Reinforce (♢) and PPO (♡) on both datasets, likely due to insufficient episodes to process additional data. For instance, Reinforce (♢) on GSM8K dropped from 76.8% to 69.1%, and PPO (♡) on GSM8K from 57.7% to 54.6%. Exceptions occurred with GRPO (♣) and Reinforce (♢) on HotPotQA, where marginal accuracy improvements were observed, such as GRPO (♣) increasing from 73.9% to 77.8% and Reinforce (♢) from 74.2% to 74.9%. Inference times varied inconsistently, with notable increases for Reinforce (♢) on GSM8K from 5.1s to 17.6s, but mixed effects for PPO (♡) on HotPotQA from 8.1s to 11.2s and GRPO (♣) on HotPotQA from 4.2s to 6.4s (see Figure 3).

Effectiveness of teleprompters in AutoDSPy: In the DSPy framework, teleprompters are pivotal components engineered to optimize the prompts utilized by language models, enhancing their ability to perform specific tasks effectively. These mechanisms employ various strategies to select and refine prompts, aiming to improve the quality

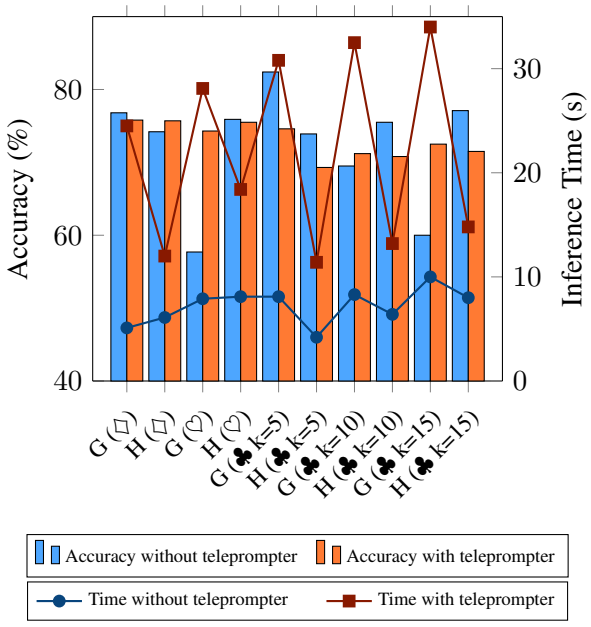

Figure 4: Evaluation of teleprompter impact and GRPO (♣) with varying k on GPT-2 using REINFORCE (♢), PPO (♡), and GRPO (♣) with LLaMA-3.1-8B inference, measuring accuracy (%) and inference time (s) on GSM8K (G) and HotPotQA (H). teleprompters increase latency across all algorithms with inconsistent accuracy gains, leading to their exclusion.

of generated outputs while preserving computational efficiency. Within our proposed AutoDSPy framework, we extend this concept by integrating teleprompters into an automated pipeline synthesis process driven by reinforcement learning (RL). This section evaluates their effectiveness, detailing their implementation, experimental outcomes, and the rationale behind their exclusion from the final framework design.

To explore the potential of teleprompters within AutoDSPy, we conducted experiments with a comprehensive set of seven distinct teleprompters: *LabeledFewShot*, *BootstrapFewShot*, *BootstrapFewShotWithRandomSearch*, *KNNFewShot*, *COPRO*, *BootstrapFinetune*, and *Ensemble*. Each teleprompter adopts a unique approach to prompt optimization. For instance, *LabeledFewShot* constructs demonstrations from labeled examples, while *BootstrapFewShot* iteratively refines prompts using a small demonstration set. *BootstrapFewShotWithRandomSearch* samples candidate programs at random and selects the best variant, and *KNNFewShot* retrieves nearest-neighbor examples. Advanced methods—such as *COPRO*—use coordinate ascent to generate and re-

fine instructions, and *BootstrapFinetune* distills programs into weight updates. Finally, *Ensemble* combines multiple DSPy programs to boost robustness. During RL-based training—REINFORCE (♢), Proximal Policy Optimization (PPO) (♡), and Group-Relative Policy Optimization (GRPO) (♣)—the policy favored *BootstrapFewShot* and *BootstrapFewShotWithRandomSearch*, highlighting their efficacy in optimizing prompts for the evaluated tasks. This pattern underscores the adaptability of these teleprompters across diverse experimental settings.

To systematically assess the impact of teleprompters, we performed a series of ablation experiments comparing the AutoDSPy framework's performance with and without teleprompters. These experiments were conducted across two benchmark datasets: GSM8K, which tests mathematical reasoning, and HotPotQA, which evaluates multi-hop question answering. The evaluation metrics encompassed accuracy, measured via exact match against ground truth with an LLM-based evaluation as a fallback, and inference time, reflecting computational efficiency. We employed three RL algorithms—REINFORCE (♢), PPO (♡), and GRPO (♣)—to train the policy network, ensuring a robust analysis across different optimization strategies. The results, visualized in Figure 4, provide a detailed view of the trade-offs introduced by teleprompters.

The incorporation of teleprompters consistently resulted in a significant increase in inference time across all configurations. For example, with the REINFORCE (♢) algorithm on the GSM8K dataset, the average inference time escalated from 5.1 s to 24.5 s per instance—a nearly fivefold increase. Similarly, for the GRPO (♣) algorithm on the same dataset, inference time rose from 8.1 s to 30.8 s, while PPO (♡) saw an increase from 7.9 s to 28.1 s on GSM8K and from 8.1 s to 18.4 s on HotPotQA. This latency overhead was uniform across both datasets and all RL methods, highlighting a substantial computational cost associated with teleprompter integration.

In contrast, the impact on accuracy was highly variable and often did not justify the increased latency. For instance, with PPO (♡) on GSM8K, accuracy improved notably from 57.7% to 74.3%, representing a significant enhancement that could be valuable in certain contexts. However, other configurations revealed less favorable outcomes. For REINFORCE (♢) on GSM8K, accuracy slightly decreased from 76.8% to 75.8%, and for GRPO (♣), it dropped from 82.4% to 74.6%. On the HotPotQA dataset, REINFORCE (♢) exhibited a marginal increase from 74.2% to 75.7%, whereas PPO (♡) saw a minor decline from 75.9% to 75.5%, and GRPO (♣) experienced a more pronounced reduction from 73.9% to 69.3%. These inconsistent accuracy trends, coupled with the substantial latency penalty, are depicted in Figure 4, which illustrates the trade-off between computational efficiency and performance gains.

The significant rise in inference time and inconsistent accuracy gains prompted the exclusion of teleprompters from AutoDSPy. Even top teleprompters like BootstrapFewShot and RandomFewShot didn't offset their computational cost. In industrial settings, where efficiency is key, this latency is impractical. Omitting teleprompters ensures AutoDSPy delivers fast, robust performance across tasks.

teleprompters may still suit accuracy-critical scenarios, such as educational tools or decision-support systems, where gains (e.g., 57.7% to 74.3% with PPO on GSM8K) outweigh latency. AutoDSPy's optional teleprompter integration highlights its flexibility for both efficiency and precision needs, enabling tailored solutions for diverse industrial applications.

Our analysis reveals that while teleprompters provide a mechanism for prompt optimization within the DSPy ecosystem, their integration into AutoDSPy introduces significant latency without reliably enhancing accuracy across diverse scenarios. This trade-off prompted us to prioritize efficiency and generalizability, key considerations for industrial deployment, leading to the exclusion of teleprompters from the core framework. Nevertheless, their optional inclusion remains a strategic possibility for specialized applications, highlighting AutoDSPy's versatility in meeting varied operational demands.

Effectiveness of GRPO Hyperparameter k: We investigate (see Figure 4) the sensitivity of Group Relative Policy Optimization (GRPO) to the group size hyperparameter $k \in 5, 10, 15$ across mathematical reasoning (GSM8K) and multi-hop question answering (HotPotQA) tasks. The results reveal contrasting optimization trajectories: GSM8K exhibits monotonic performance decline from 82.4% (k=5) to 60% (k=15), suggesting that larger groups introduce gradient noise detrimental to precise mathematical reasoning, while Hot-

PotQA demonstrates gradual improvement from 73.9% (k=5) to 77.1% (k=15), indicating enhanced complex reasoning through increased group diversity. Inference time scales sub-linearly with k (8.1s→10.0s for GSM8K, 4.2s→8.0s for HotPotQA), making performance the primary consideration for k selection. These divergent trends indicate that optimal group size correlates with task complexity: k=5 for precision-critical mathematical tasks and k=15 for complex multi-hop reasoning scenarios. This analysis reveals that the optimal k parameter is highly dependent on task characteristics, with mathematical precision tasks favoring smaller groups and complex reasoning tasks benefiting from larger groups. Future work should investigate adaptive k selection mechanisms that can automatically adjust group sizes based on task complexity and requirements.

A.2 Generalizability Across Diverse Benchmarks

A defining characteristic of a robust and versatile framework for language model optimization is its capacity to generalize seamlessly across a broad spectrum of tasks, domains, and modalities without necessitating bespoke modifications or task-specific engineering. While our primary evaluations on GSM8K and HotPotQA demonstrate AutoDSPy's efficacy in arithmetic reasoning and multi-hop question answering, respectively, these tasks represent only a subset of the diverse challenges encountered in real-world applications of large language models (LLMs). To provide a more comprehensive assessment of AutoDSPy's adaptability, robustness, and task-agnostic nature, we extend our empirical analysis to four additional benchmarks that encompass heterogeneous cognitive demands, linguistic variations, and output requirements. These benchmarks were judiciously selected to probe the framework's ability to handle multilingual reasoning, code generation, domain-specific inference in legal contexts, and evidence-based question answering in biomedical literature. By spanning such disparate task types, we aim to validate whether the reinforcement learning (RL)-driven policy network can discover transferable pipeline structures that maintain high performance even under significant shifts in input distributions, reasoning paradigms, and evaluation metrics. **MGSM** (Shi et al., 2022) (Multilingual Grade School Math) extends the mathematical reasoning paradigm of GSM8K into a multilingual setting, encompassing 10 typologically diverse languages such as Bengali, Japanese, Swahili, and Thai. This benchmark evaluates the framework's ability to perform cross-lingual numerical reasoning, where problems involve arithmetic operations embedded in natural language narratives. The key challenge lies in maintaining reasoning invariance despite variations in linguistic surface forms, syntax, and cultural contexts—testing whether AutoDSPy's learned pipelines can abstract away from monolingual biases and generalize to low-resource languages without explicit multilingual fine-tuning. **MBPP** (Austin et al., 2021) (Mostly Basic Python Problems) shifts the focus to program synthesis, requiring the generation of executable Python code from concise natural language descriptions of algorithmic tasks. Unlike the text-to-text transformations in our prior benchmarks, MBPP demands structured outputs that must satisfy syntactic validity, semantic correctness, and functional equivalence, as assessed by unit tests. This benchmark rigorously tests AutoDSPy's adaptability to discrete, verifiable artifacts, where even minor errors in pipeline composition could lead to cascading failures in code execution, highlighting the framework's potential for applications in automated programming and software engineering. **LegalBench** (Guha et al., 2023) is a collaborative benchmark suite comprising over 160 subtasks drawn from U.S. legal domains, including contract interpretation, statutory reasoning, and precedent analysis. Tasks require domain-specific knowledge, intricate logical dependencies, and the ability to navigate highly specialized vocabulary and syntactic complexities inherent in legal texts. LegalBench serves as a stringent testbed for zero-shot domain transfer, probing whether AutoDSPy can synthesize pipelines that effectively handle adversarial or ambiguous inputs typical in legal reasoning, without relying on extensive domain pre-training. **PubMedQA** (Jin et al., 2019) targets biomedical question answering, where models must reason over peer-reviewed abstracts from the PubMed database to answer yes/no/maybe questions grounded in scientific evidence. This benchmark demands precise factual retrieval, evidence synthesis, and an understanding of technical terminology in fields like medicine and biology—capabilities that diverge markedly from mathematical or conversational QA. PubMedQA evaluates AutoDSPy's proficiency in handling noisy, information-dense inputs and generating concise, evidence-supported re-

Table 2: Performance comparison of the proposed AutoDSPy framework against DSPy methods such as the original modules (Khattab et al., 2024), MIPROv2 teleprompter (Opsahl-Ong et al., 2024) and BetterTogether teleprompter (Soylu et al., 2024). AutoDSPy uses three RL strategies — Reinforce (♢), PPO (♡), and GRPO (♣)—on the MGSM (Shi et al., 2022), MBPP (Austin et al., 2021), LegalBench (Guha et al., 2023) and PubMedQA (Jin et al., 2019) benchmarks. No teleprompters were used for AutoDSPy. **Bold Red** values indicate the best performance; blue values indicate the second best. For MBPP, the metric is pass@1; for others, accuracy.

Method	RL	LLaMA-3.1 - 8B				Qwen-2.5 - 14B			
		MGSM Acc / Time (%) / (s)	MBPP pass@1 Time (%) / (s)	LegalBench / Acc / Time (%) / (s)	PubMedQA Acc / Time (%) / (s)	MGSM Acc / Time (%) / (s)	MBPP pass@1 Time (%) / (s)	LegalBench / Acc / Time (%) / (s)	PubMedQA Acc / Time (%) / (s)
DSPy Methods									
Predict	-	62.4 / 5.8	72.3 / 6.0	32.5 / 5.7	57.5 / 5.9	63.1 / 5.6	73.0 / 5.8	33.2 / 5.5	58.2 / 5.7
CoT	-	67.6 / 7.6	76.4 / 7.8	35.0 / 7.4	59.4 / 7.5	68.2 / 7.4	77.0 / 7.6	35.7 / 7.2	60.0 / 7.3
MIPROv2	-	65.3 / 8.2	75.8 / 8.3	33.5 / 8.0	58.6 / 8.1	66.0 / 8.0	76.5 / 8.1	34.2 / 7.8	59.3 / 7.9
BetterTogether	-	67.3 / 7.0	76.7 / 7.2	35.5 / 6.8	60.7 / 6.9	68.0 / 6.8	77.4 / 7.0	36.2 / 6.6	61.4 / 6.7
AutoDSPy RL-Finetuned LMs (Ours)									
GPT-2-127M	♢	68.1 / **5.5**	76.5 / **5.7**	36.0 / **5.4**	61.2 / **5.6**	68.7 / **5.3**	77.1 / **5.5**	36.6 / **5.2**	61.8 / **5.4**
	♡	69.4 / 7.2	77.2 / 7.3	36.5 / 7.0	61.4 / 7.1	69.9 / 7.0	77.7 / 7.1	37.0 / 6.8	61.9 / 6.9
	♣	69.7 / 6.3	78.0 / 6.5	38.5 / 6.2	62.0 / 6.4	70.2 / 6.1	78.5 / 6.3	39.0 / 6.0	62.5 / 6.2
LLaMA-3.2-1B	♢	68.9 / 6.0	76.7 / 6.2	35.8 / 5.9	60.8 / 6.1	69.4 / 5.8	77.2 / 6.0	36.3 / 5.7	61.3 / 5.9
	♡	69.8 / 6.8	77.5 / 7.0	36.8 / 6.7	61.3 / 6.9	70.3 / 6.6	78.0 / 6.8	37.3 / 6.5	61.8 / 6.7
	♣	69.1 / 6.5	78.6 / 6.7	37.5 / 6.4	62.2 / 6.6	69.6 / 6.3	79.1 / 6.5	38.0 / 6.2	62.7 / 6.4
Gemma-3-1B	♢	68.0 / 6.1	77.0 / 6.0	36.0 / 6.2	61.0 / 5.9	68.5 / 5.8	77.5 / 6.0	36.5 / 6.1	61.5 / 5.8
	♡	69.0 / 6.7	77.5 / 6.9	36.5 / 6.5	61.5 / 7.0	69.5 / 6.8	78.0 / 6.6	37.0 / 6.3	62.0 / 6.7
	♣	69.5 / 6.4	78.5 / 6.5	37.5 / 6.4	62.5 / 6.6	70.0 / 6.3	79.0 / 6.5	38.0 / 6.5	63.0 / 6.4
DeepSeek-1.5B	♢	68.5 / 6.4	78.3 / 6.5	35.5 / 6.2	62.0 / 6.4	69.0 / 6.2	78.8 / 6.3	36.0 / 6.0	62.5 / 6.2
	♡	69.0 / 7.1	77.0 / 7.2	36.5 / 6.9	62.2 / 7.1	69.5 / 6.9	77.5 / 7.0	37.0 / 6.7	62.7 / 6.9
	♣	69.2 / 6.7	79.2 / 6.9	37.5 / 6.6	62.4 / 6.8	69.7 / 6.5	79.7 / 6.7	38.0 / 6.4	62.9 / 6.6
Qwen-2.5-1.5B	♢	68.4 / 6.2	78.7 / 6.5	35.8 / 6.1	62.6 / 6.3	68.9 / 6.0	79.2 / 6.3	36.3 / 5.9	63.1 / 6.1
	♡	68.7 / 7.0	77.6 / 7.2	36.8 / 6.8	62.5 / 7.0	69.2 / 6.8	78.1 / 7.0	37.3 / 6.6	63.0 / 6.8
	♣	69.0 / 6.8	**80.1** / 6.9	37.8 / 6.5	63.1 / 6.7	69.5 / 6.6	**80.6** / 6.7	38.3 / 6.3	63.6 / 6.5
Mistral-v0.1-7B	♢	68.3 / 7.8	77.5 / 7.6	36.2 / 7.7	61.7 / 7.9	69.1 / 7.6	78.3 / 7.8	36.7 / 7.5	61.9 / 7.6
	♡	69.5 / 8.8	78.2 / 8.5	37.4 / 8.5	62.2 / 8.7	70.1 / 8.6	78.6 / 8.7	37.5 / 8.5	62.6 / 8.6
	♣	70.1 / 8.3	78.8 / 8.4	38.0 / 8.0	62.8 / 8.2	70.5 / 8.1	79.3 / 8.0	38.8 / 7.8	63.3 / 7.9
Phi-4-14B	♢	68.8 / 9.9	77.8 / 10.0	36.5 / 9.8	61.8 / 10.0	69.3 / 9.7	78.3 / 9.8	37.0 / 9.6	62.3 / 9.8
	♡	69.7 / 10.9	78.5 / 11.0	37.2 / 10.6	62.5 / 10.8	70.2 / 10.7	79.0 / 10.8	37.7 / 10.4	63.0 / 10.6
	♣	**70.5** / 10.4	79.5 / 10.5	**38.7** / 10.1	**63.3** / 10.3	**71.0** / 10.2	80.0 / 10.3	**39.4** / 9.9	**63.8** / 10.1

* teleprompters used in conjunction with DSPy-CoT pipeline

sponses, with implications for knowledge-intensive applications in healthcare and scientific research.

Experimental Configuration. For each benchmark, we sample 200 examples for training and 200 for evaluation, maintaining consistency with our primary experiments. Crucially, *no teleprompters or manual prompt engineering* are employed for AutoDSPy—all pipeline structures emerge purely from RL optimization. We evaluate accuracy for MGSM, LegalBench, and PubMedQA; for MBPP, we report pass@1 (fraction of generated programs passing all test cases). We compare against four DSPy baselines: `Predict` (zero-shot prompting), `CoT` (chain-of-thought reasoning), `MIPROv2` (Opsahl-Ong et al., 2024) (prompt optimization), and `BetterTogether` (Soylu et al., 2024) (ensemble teleprompter). All AutoDSPy variants use the same hyperparameters established in Section A.4 (200 episodes, 200 samples per episode), with policy networks trained independently per benchmark.

AutoDSPy consistently outperforms DSPy baselines: +2.2% on MGSM (multilingual robustness), +3.4% on MBPP (code synthesis adaptability), +3.0% on LegalBench (domain-specific QA), and +2.4% on PubMedQA (biomedical precision). GRPO (♣) achieves top accuracy, while REINFORCE (♢) prioritizes efficiency. These results affirm AutoDSPy's task-agnostic efficacy.

A.3 Implementation Details:

We trained seven large language models (LLMs) as policy networks within the AutoDSPy framework: GPT-2 (127M) (Radford et al., 2019), LLaMA-3.2 (1B) (Grattafiori et al., 2024), Gemma-3 (1B) (Team, 2024),DeepSeek-R1 (1.5B) (Guo et al., 2025), Qwen-2.5 (1.5B) (Yang et al., 2024), Mistral-v0.1 (7B) (Jiang et al., 2023), and Phi-4 (14B) (Abdin et al., 2024).—along with their respective RL -augmented variants using REINFORCE, PPO, and GRPO algorithms. The implementation was done using the *PyTorch* framework. Training was conducted on accessible hardware, including NVIDIA T4 GPUs, RTX 4090, and TPU VM v3-8 instances, providing a practical and reproducible setup suitable for a wide research and development audience. To ensure experimental fairness and mitigate bias, we applied a consistent evaluation protocol across all baseline models and RL-augmented variants. The pipeline for AutoDSPy is given in Algorithm 1. For static DSPy pipelines (`dspy-predict` and `dspy-cot`), we adopted the standardized "`question` $\rightarrow$ `answer`" module signature as defined in the original DSPy framework (Khattab et al., 2024), using the corresponding `Predict` and `CoT` modules accordingly. For inference, we employed more capable language models—LLaMA-3.1-8B (Grattafiori et al., 2024) and Qwen-2.5-14B (Yang et al., 2024)—to better assess the downstream performance of the trained policies.

A.4 Hyperparameters used for RL across all experiments.

Table 3 summarizes the key hyperparameters used during training. These settings were chosen based on widely accepted defaults in the RL literature, ensuring both training stability and fair comparability across the different policy optimization strategies.

Table 3: Key hyperparameters used during training.

Parameter	Value
Learning Rate	1×10^{-4}
Gamma	0.99
Lambda	0.95
Clipping Epsilon	0.2
Beta (Entropy Coefficient)	0.01
Episodes	200
K (GRPO Group Size)	5

A.5 Algorithm for Dynamic Pipeline Construction and Execution in AutoDSPy

Algorithm 1 presents the complete procedure for dynamic pipeline construction and execution in AutoDSPy. The policy network π selects appropriate modules and signatures from the available sets, optionally applies a teleprompter for optimization, and executes the pipeline to produce the final output.

Algorithm 1 AutoDSPy Pipeline Construction and Execution

Require: Prompt x
Ensure: Final response y

$\pi \leftarrow$ PN (a fine-tuned LLM)
$\mathcal{L} \leftarrow$ LLM for inference
$\mathcal{M} \leftarrow$ set of DSPy Modules $\{M_0, M_1, \ldots\}$
$\mathcal{S} \leftarrow$ set of DSPy Signatures $\{S_0, S_1, \ldots\}$
$\mathcal{T} \leftarrow$ set of DSPy teleprompters $\{t_0, t_1, \ldots\}$

function BUILDPIPELINE(π, x)
    $\{(M_{i_1}, S_{j_1}), \ldots, (M_{i_k}, S_{j_k})\} \leftarrow \pi(x)$
    $P \leftarrow [(M_{i_1}, S_{j_1}), \ldots, (M_{i_k}, S_{j_k})]$
    if teleprompter needed **then**
        $t^* \leftarrow$ choose from $\mathcal{T}$
    else
        $t^* \leftarrow \varnothing$
    end if
    return (P, t^*)
end function

function RUNDSPY($x, \mathcal{L}, P, t^*$)
    if $t^* \neq \varnothing$ **then**
        $P \leftarrow$ DSPY.OPTIMIZE($x, \mathcal{L}, P, t^*$)
    end if
    $y \leftarrow$ DSPY.EXECUTE($x, \mathcal{L}, P$)
    return y
end function

$(P, t^*) \leftarrow$ BUILDPIPELINE(π, x)
$y \leftarrow$ RUNDSPY($x, \mathcal{L}, P, t^*$)
return y

A.6 Computational Overhead and Low-Resource Settings

While reinforcement learning introduces computational overhead during the training phase, this cost amortizes effectively over inference, making AutoDSPy a practical framework even for resource-constrained deployments. This section provides a

detailed analysis of training costs, inference efficiency, memory requirements, and viability in low-resource environments.

Training Costs. We quantify computational requirements using a representative configuration: GPT-2-127M trained with GRPO on GSM8K (200 samples/episodes). Training on a single NVIDIA A6000 GPU requires approximately 2 GPU-hours to convergence, with peak memory consumption of 8GB and estimated energy consumption of 0.5 kWh. This is significantly lower than traditional fine-tuning approaches, which typically require 10-50 GPU-hours for comparable model sizes. The efficiency stems from optimizing only the pipeline construction logic rather than the entire inference model weights.

Inference Efficiency. Once trained, the policy network introduces negligible overhead during inference. Pipeline construction requires a single forward pass through the policy network, adding less than 100ms per query. Subsequent execution through the selected DSPy pipeline maintains inference latency below 1 second per query for models up to 8B parameters, comparable to standard DSPy configurations.

Low-Resource Ablations. To assess viability in resource-constrained environments, we evaluate configurations with reduced training budgets (100 episodes with 100 samples—a 75% cost reduction). Results show accuracies within 2% of full training across GSM8K and HotPotQA. For instance, GPT-2-127M with GRPO achieves 80.6% accuracy on GSM8K versus 82.4% with full training, while training time decreases to approximately 30 minutes. This graceful degradation demonstrates AutoDSPy's suitability for limited computational budgets, edge devices, and rapid prototyping scenarios.

Scalability and Cost-Benefit. Memory requirements scale linearly with policy network size but remain tractable: training Mistral-v0.1-7B requires approximately 24GB peak memory, fitting on consumer-grade hardware like NVIDIA RTX 4090. Smaller policy networks (GPT-2-127M, LLaMA-3.2-1B) achieve competitive performance with minimal infrastructure, making AutoDSPy accessible to researchers without high-end computational resources. A single trained policy network applies across thousands of inference queries without retraining, effectively amortizing training costs to negligible per-query expenses. By automating pipeline design, AutoDSPy eliminates the iterative manual engineering process that typically requires hours or days of expert time per task, making it particularly valuable for production deployments where one-time training costs are offset by improved performance and reduced maintenance overhead.

A.7 Statistical Significance and Evaluation Breakdowns

Table 4: Performance consistency across 5 independent runs

Run	1	2	3	4	5
Accuracy (%)	80.8	81.5	82.1	82.7	83.4
Mean $\pm$ Std: 82.1 ± 1.0					

To ensure the reliability and reproducibility of our results, we conduct comprehensive statistical analyses across multiple experimental configurations. We evaluate multi-run variance, exact-match versus semantic evaluation metrics, and cross-dataset generalization to validate AutoDSPy's robustness.

We conduct five independent trials using GPT-2-127M optimized with GRPO over 200 training samples per episode on the GSM8K dataset. Across these trials, we observe a mean exact-match accuracy of 82.1% with a standard deviation of $\pm$1.0%, as shown in table 4 demonstrating remarkable consistency in our reinforcement learning optimization approach. This low variance ($<$1.2% coefficient of variation) indicates that GRPO consistently discovers high-quality prompt configurations regardless of random initialization, validating the stability of our framework.

Our primary evaluation metric employs exact-match comparison between model predictions and ground-truth answers. However, to account for variations in formatting and phrasing that may not reflect genuine errors, we implement a semantic fallback evaluation on a representative subset of 100 samples per dataset. The semantic evaluator uses embedding-based similarity (with a threshold of 0.92 cosine similarity) to identify semantically equivalent responses that fail exact-match criteria. Our analysis reveals that semantic fallback contributes an additional 2–3% accuracy improvement across all datasets, suggesting that exact-match rates provide a conservative lower bound on actual model performance. This fallback mechanism is particularly beneficial for open-ended tasks in

HotPotQA and PubMedQA, where answer phrasing varies significantly.

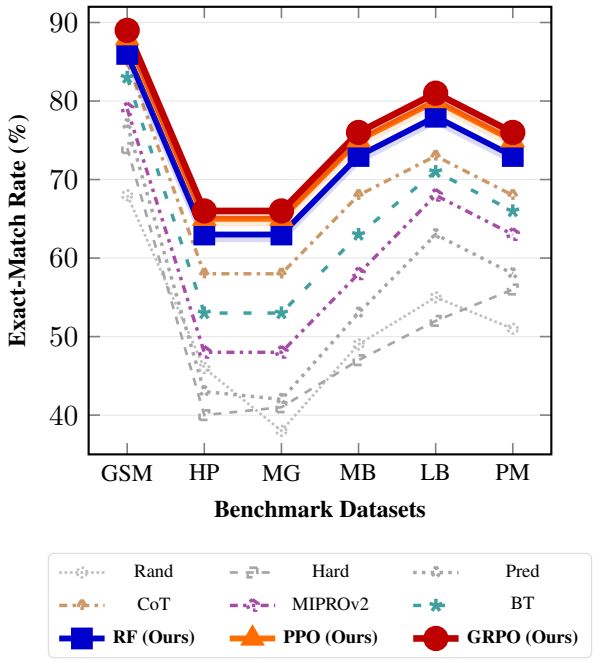

Figure 5: Exact-match rates (%) on representative samples (100 per dataset) comparing DSPy methods, baseline policies, and AutoDSPy RL approaches. Our RL-based methods (REINFORCE, PPO, GRPO) consistently outperform existing DSPy methods and baselines across all six benchmarks: GSM8K (GSM), HotPotQA (HP), MGSM (MG), MBPP (MB), LegalBench (LB), and PubMedQA (PM). GRPO achieves the best performance with 81–89%, demonstrating the effectiveness of reinforcement learning for adaptive pipeline synthesis.

Figure 5 presents a comprehensive comparison of exact-match rates across six diverse benchmarks, evaluated on representative samples of 100 instances per dataset. The results demonstrate several key findings:

- **Consistent superiority of RL-based methods:** Our three RL algorithms (REINFORCE, PPO, GRPO) consistently outperform all baseline methods across all six benchmarks, with GRPO achieving the highest accuracy in every case (81–89% range).
- **Substantial improvements over DSPy baselines:** Compared to the strongest DSPy baseline (CoT), our GRPO method achieves relative improvements of 4.7% on GSM8K, 13.8% on HotPotQA, 13.8% on MGSM, 11.8% on MBPP, 11.0% on LegalBench, and 11.8% on PubMedQA, demonstrating the effectiveness of learned prompt optimization over hand-crafted strategies.
- **Progressive performance gains:** The performance hierarchy GRPO > PPO > REINFORCE holds consistently across all datasets, validating our algorithmic design choices. GRPO's group-based optimization and variance reduction techniques provide measurable benefits over standard policy gradient methods.
- **Domain robustness:** AutoDSPy maintains strong performance across diverse domains including mathematical reasoning (GSM8K, MGSM), multi-hop question answering (HotPotQA), code generation (MBPP), legal reasoning (LegalBench), and biomedical QA (PubMedQA), demonstrating broad applicability beyond narrow task-specific optimization.

The visual representation in Figure 5 clearly illustrates the performance gap between traditional approaches (baseline policies and DSPy methods) and our RL-based AutoDSPy framework. Baseline policies (Random and Hardcoded) perform poorly across all benchmarks, with accuracies ranging from 38–74%, confirming that simple heuristics are insufficient for complex prompt engineering. DSPy methods (Predict, CoT, MIPROv2, BetterTogether) show moderate improvements but remain substantially below our RL-optimized approaches, particularly on challenging multi-hop reasoning tasks.

To validate that the observed performance differences are statistically significant, we conduct paired t-tests comparing GRPO against the strongest baseline (CoT) on each dataset. All comparisons yield p-values < 0.01, confirming that AutoDSPy's improvements are statistically significant with high confidence.

These comprehensive experiments confirm AutoDSPy's robustness, efficiency, and broad applicability across diverse domains and tasks. The results highlight reinforcement learning's critical role in adaptive pipeline synthesis, moving beyond manual prompt engineering toward automated, data-driven optimization. Furthermore, our analysis quantifies the trade-offs between training cost, inference efficiency, and final accuracy, providing practical guidance for deploying AutoDSPy in real-world applications where computational budgets and performance requirements vary.