VALUE AUGMENTED SAMPLING: PREDICT YOUR REWARDS TO ALIGN LANGUAGE MODELS

Idan Shenfeld*[1], Seungwook Han*[1], Akash Srivastava[2], Yoon Kim[1] , Pulkit Agrawal[1]

ABSTRACT

With Large Language Models (LLMs) learning a plethora of behavior from Internet data, it has become ever more important to adapt and align these models to cater to different human preferences, learn new skills, and unlearn harmful behavior. Currently, there exists a dichotomy of solutions, each with its own set of advantages and disadvantages. Search-based methods, such as Best-of-N and Monte Carlo Tree Search, are effective but expensive during inference and, therefore, infeasible in practice. RL-based methods, such as Proximal Policy Optimization, are computationally efficient, but are not competitive in performance. To this end, we propose a novel framework for reward optimization, Value Augmented Sampling (VAS), to effectively adapt LLMs at inference-time, while significantly reducing the computational cost compared to existing search-based methods. At the heart of our method lies the idea of augmenting the LLM's output distribution with expected reward estimates to guide the generation process toward high-reward responses at the token level. Our method outperforms established baselines, such as PPO and DPO, in improving summarization and multi-turn chat dialogue and achieves comparable results to Best-of-128 with lower inference costs. We also demonstrate the capability to align a closed-source, proprietary model (such as GPT-3.5) on learning to use a new API tool.

1 INTRODUCTION

Large language models (LLMs) trained on large amounts of Internet data are powerful models for many applications, such as programming and web-based search (Schick et al., 2023; Rozière et al., 2024; Chiang et al., 2023). Usually, once the models are pre-trained, they are adapted to improve their task-specific capabilities, suppress unwanted behavior, or change their response style. A popular approach for adaptation is to define any target task or behavior as a reward function and to leverage one of many search-based or optimization-based frameworks to maximize the expected reward.

Each of these methods brings its own set of advantages and disadvantages. In this work, we first identify four key characteristics that essential for a practical adaptation method. First of all, an adaptation method should be *performant*, effectively maximizing the specified reward. Second, the method needs to be as *computationally efficient* as possible, especially during inference. Third, a method should allow for *inference time adaptability*. Adapting LLMs requires many design choices, from choosing the tasks to deciding how much to optimize for the specific task. At times, it may be necessary to combine multiple tasks simultaneously. Thus, an ideal method should allow for control over some of these design choices without a need to retrain the LLM. Finally, an ideal method should be *flexible* to work with any model. Since many state-of-the-art models are available only via API (Google, 2024; OpenAI, 2024), we would aspire that the method can work with both open-source and closed-source models.

Search-based methods have especially touted their strength in maximizing performance. Even simple methods like Best-of-N (BoN), where one optimizes for a given reward by sampling N number of sequences from the model and choosing the one that maximizes the reward, can achieve a very high reward given a sufficiently large N. Moreover, since the search is conducted at inference time, we neither need to decide the design factors for adaptation a priori nor need access to the model's weights. However, the critical problem of these methods is their high computation cost. To take BoN for example, it costs N times the number of generations with the LLM and N times the evaluations with the reward model. Therefore, it is computationally infeasible to use in practice.

* Equal contribution, correspondences to {idanshen, swhan}@mit.edu, [1] MIT, [2] MIT-IBM Watson AI Lab

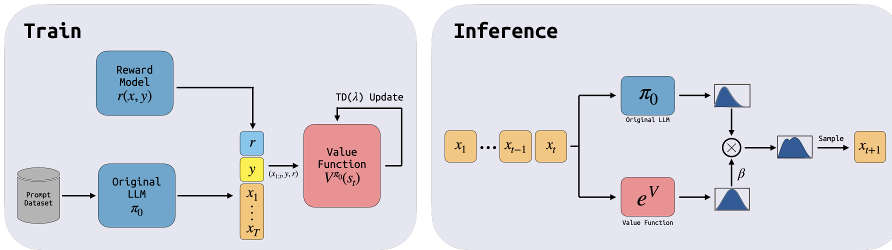

Figure 1: Overview of VAS training and inference. During training, our algorithm learns a Value function estimator using TD learning. In inference, we augment the output distribution of the original LLM, π_0, with the exponentiated Values scaled by a weight coefficient β.

To alleviate this high inference-time cost, many optimization-based approaches have been proposed and have recently gained popularity for adapting LLMs. These methods, like Proximal Policy Optimization (PPO) (Schulman et al., 2017) and its variants (Wu et al., 2023), leverage the actor-critic framework (Sutton & Barto, 2018) to learn a reward-maximizing policy, instead of conducting search. However, these learning algorithms still fail to outperform the simple, naive search method of BoN, which has been shown empirically to be the most effective by far (Gao et al., 2023). A hypothesis as to why RL-based approaches lag behind BoN in performance is their optimization instability. Actor-critic methods conduct a bi-level optimization, alternating between learning a Value estimator for the current policy and using this estimator to improve the policy. We posit that PPO does not seem to converge to the optimal solution because of the instabilities accompanying this optimization process. In addition to the performance gap, the optimization-based methods also lack inference time adaptability and require access to the models' weights. For any given design choice, the LLM must be re-trained and minimal control over the model's task-specific behavior is allowed at inference-time.

In this paper, we present an reward-maximization adaptation method to bridge the gap in performance between current optimization-based methods and Best-of-N, while reducing inference-time cost, enabling inference-time control over the adaptation process, and waiving the need for the model's weights. Instead of conducting a costly search over entire sequences as in BoN, we propose to conduct a token-level search by predicting the expected reward from partial sequences. At each step in the generation process, as an approximation to searching over all future sequences, we first estimate the expected reward associated with choosing each token. Then, we augment the LLM distribution with these values and sample the next token from it.

Our main contributions are as follows: (1) We introduce Value Augmented Sampling (VAS), a novel reward optimization algorithm, that uses a Value estimator to guide the decoding process of an LLM. Our method outperforms strong baselines, such as PPO and DPO, in improving summarization and creating a more helpful/harmless chat assistant. We further demonstrate that VAS is on par with Best-of-128 while being more computationally efficient; (2) We showcase a novel application of aligning a closed-source model. With our method, we improve GPT-3.5 capability in using new APIs.

2 Method

2.1 From Reward Optimization to Value Augmented Sampling.

Consider the following finite-time Markov Decision Process (MDP) for a generative model π ("policy"). At time step t, the policy observes a state s_t, defined as the text prompt y concatenated with the response generated so far $x_{\leq t}$, and outputs an action $x_{t+1} \in \mathbb{V}$, where $\mathbb{V}$ is the vocabulary space of the model. Once the action is chosen, the model deterministically transitions into the next state $s_{t+1} = s_t \oplus x_{t+1}$, where $\oplus$ is the concatenation operation. The episode ends either after the policy generates T new tokens or an [EOS] token. At the end of the episode, a scalar reward is collected from the reward model $r(s_T) : \mathbb{V}^T \to \mathbb{R}$ where s_T is the generated response. In the following Kullback-Leibler (KL)-regularized RL setting (Ng et al., 2000; Boularias et al., 2011), we aim to learn a policy that maximizes the expected reward while keeping the distance from the initial π_0 under a threshold ϵ:

$$\max_{\pi} \mathbb{E}_{s_T \sim \pi}[r(s_T)] \quad s.t \quad D_{KL}(\pi || \pi_0) < \epsilon \tag{1}$$

Prior works have tackled Equation 1 by directly optimizing a language model to be the solution π, which is known to be complex and unstable. As an alternative, we look into the following closed-form solution to the KL-regularized RL problem (Peters & Schaal, 2007; Peng et al., 2019).

Proposition 2.1. *Let the expected reward of being in state s_t, and choosing token x_{t+1} be the Q-value $Q^{\pi_0}(x_{t+1}|s_t) = \mathbb{E}_{\mathsf{s_T}\sim\pi_0}[r(\mathsf{s_T})|\mathsf{s}_t = s_t, \mathsf{x}_{t+1} = x_{t+1}]$. The solution to the optimization problem in Equation 1 is:*

$$\pi(x_{t+1}|s_t) \propto \pi_0(x_{t+1}|s_t)e^{\beta Q^{\pi_0}(x_{t+1}|s_t)} \tag{2}$$

β is a scalar dependent on ϵ and independent of x_{t+1}.

This proposition hints that instead of learning π, it can be sufficient to only learn Q^{π_0}. Since we have access to $\pi_0(x_{t+1}|s_t)$, we can estimate the optimal solution without having an explicit model for π. Learning a Q-value estimator for a given policy is a well-studied problem (Sutton & Barto, 2018). However, the action space of language models is the set of all possible next tokens $\mathbb{V}$, which is usually in the tens of thousands, much larger than most RL environments. Existing algorithms for Q-value estimation have been shown to be ineffective in such large discrete action spaces (Dulac-Arnold et al., 2015). To overcome this challenge, we take advantage of another unique property of the language generation task: *deterministic transitions*. Given the current state and the next action, the next state is simply the concatenation of the generated tokens. This property allows us to draw the equivalence between the Q-value of a state-action pair and the Value function and to re-write Equation 2:

$$Q^{\pi_0}(x_{t+1}|s_t) = V^{\pi_0}(x_{t+1} \oplus s_t) \rightarrow \pi(x_{t+1}|s_t) \propto \pi_0(x_{t+1}|s_t)e^{\beta V^{\pi_0}(x_{t+1}\oplus s_t)} \tag{3}$$

As seen in Equation 3, a Value function does not depend on the action space size. Therefore, when the action space is large, it can be learned more easily than a Q-value function. The trade-off of this lower learning complexity is that the Value estimator needs to predict the expected return over the entire vocabulary set $\mathbb{V}$.

2.2 Value Augmented Sampling (VAS)

Equation 3 presents an elegant but infeasible solution to the KL-regularized RL problem. To realize it into a practical algorithm, we propose the following approximation. At every decoding step, we first obtain k tokens with the highest output probabilities under our base policy π_0. Then, we calculate the Value estimation for only these k tokens and sample the subsequent action according to the following distribution:

$$\pi(x_{t+1}|s_t) \propto \begin{cases} \pi_0(x_{t+1}|s_t)e^{\beta V^{\pi_0}(x_{t+1}\oplus s_t)} & x_t \in \text{top-k} \\ \pi_0(x_{t+1}|s_t)e^{\beta \bar{V}(s_t)} & x_t \notin \text{top-k} \end{cases} \tag{4}$$

With $\bar{V}(s_t) = \frac{1}{k}\sum_{x_t\in\text{top-k}} V^{\pi_0}(x_{t+1} \oplus s_t)$. We have found that even when $k << |\mathbb{V}|$, our proposed approximation is effective, while reducing computational complexity significantly. To train our Value function, we first collect a dataset $D = \{s^i, y^i, r^i\}_i^N$ by sampling $s^i \sim \pi_0(s_T|y^i)$ and evaluating them with the reward function $r^i = r(s^i)$. Then, we use TD(λ) algorithm (Sutton & Barto, 2018; Schulman et al., 2015) to train our Value function. Our Value function is parametrized as a separate neural network, decoupled from π_0. This parametrization removes both the need of having access to the initial policy's weights and updating them. We provide an illustrative overview of our algorithm in Figure 1.

3 Experiments

We now evaluate VAS on a range of text generation tasks to show its effectiveness in aligning LLMs and highlight its unique advantages. Additional results can be found in Appendix B and E.

3.1 Summarization

Task and Experimental Details We begin with the task of improving summary generation (Ouyang et al., 2022; Wu et al., 2021; Ramé et al., 2023). For this task, we use the SEAHORSE dataset (Clark et al., 2023). This dataset contains 96K pairs of texts, summarizations, and human labels along multiple dimensions of quality. We focus on three measures: (1) **Attribution**: whether the information in the summary is fully attributable to the source article, (2) **Main ideas**: whether the summary captures the main idea(s), and (3) **Conciseness**: whether the summary is concise.

We supervised fine-tune a LLaMA-7B (Touvron et al., 2023) model on the SEAHORSE dataset. This model will serve as the initial LLM, π_0, and will be called the supervised fine-tuned (SFT) policy. Because off-the-shelf reward models are not available for this dataset, we train our own reward models with Flan-T5-L (780M parameters, (Wei et al., 2021)) to predict the human labels along the three attributes. For the Value estimator, we also use the LLaMA-7B model. We compare VAS against PPO and FUDGE (See description in Appendix D) by analyzing both the KL vs. reward and KL vs. win rate. In addition, we perform GPT-4 evaluations, where we compare against BoN and MCTS.

Results As illustrated in the left two columns of Figure 2, we outperform PPO and FUDGE on Attribution and Main Ideas. We maximize the rewards given any KL divergence constraint better and achieve better overall summarizations (as evaluated by GPT-4). On the last axis of Conciseness, although PPO achieved higher rewards than VAS, its generated summaries are judged to be worse by GPT-4. This is a tangible example of 'reward hacking' (Gao et al., 2023). We further evaluate the model's summarization ability by conducting head-to-head evaluations of generations from different models with GPT-4 and VAS approximately matches the performance of Best-of-128, a very strong but computationally expensive method. We present detailed results in Table 3 of Appendix E.

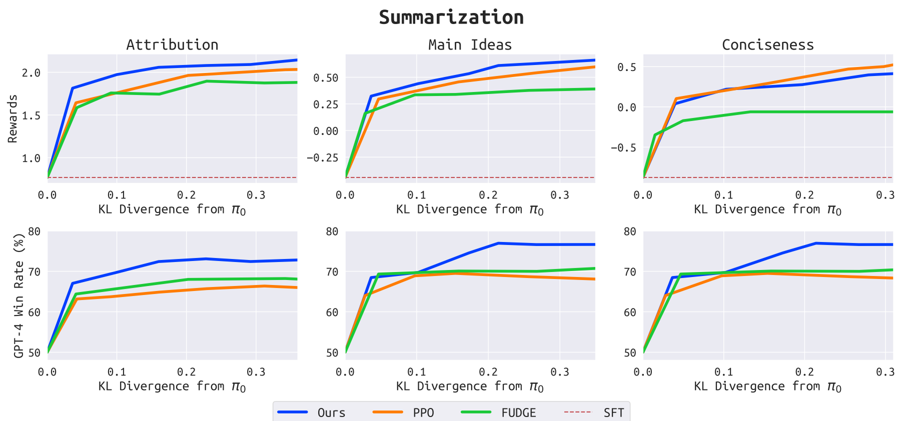

Figure 2: The KL-performance trade-off for both reward and win rate over SFT, for different summarization objectives. VAS outperforms the baselines in both Attribution and Main Ideas. In Conciseness, PPO achieves a higher reward but a significantly lower win rate. These results show that VAS maximizes the reward better without reward hacking as much as PPO.

3.2 Helpful/Harmless Chat Assistant

Task and Experimental Details The second task is to improve the LLM's capability as a helpful and harmless chat assistant. To this end, we use Anthropic's Helpfulness and Harmlessness (HH) dataset (Bai et al., 2022), which contains ∼161K examples of conversations between a human and an assistant. Each conversation is paired with two responses, one of which is labeled as preferred. The preference is based on which is more informative, honest, and safe.

We supervised fine-tune LLaMA-2-7B (Touvron et al., 2023) model on the preferred responses. On top of PPO and FUDGE, we add another strong baseline, Direct Policy Optimization (DPO) (Rafailov et al., 2023). For FUDGE, because it is not designed for pair-wise preference learning, we create a variation of FUDGE ("Online FUDGE") as a baseline for this task. Instead of learning a prefix classifier from offline data, Online FUDGE learns a Value estimator from online data and reward function by regarding it as a regression task. Therefore, it can also be seen as a TD(1) version of VAS. For GPT-4 evaluations, we additionally evaluate against Best-of-128. For the reward model, we follow (Ramé et al., 2023) and use a pre-trained reward model based on DeBERTa V3 Large (304M parameters).

Results As illustrated in the left plot of Figure 3, VAS attains the best KL-reward trade-off. DPO's weak performance on this metric is because it does not learn to maximize the reward, but learns directly from the data. On the right, we show head-to-head win rate comparisons against different models. Noticeably, we outperform DPO, a strong baseline tailored for preference learning, with a 55% win rate. Further, we show that VAS is comparable, if not slightly better, to Best-of-128. Regarding the poor performance of PPO, we report the best performance we were able to achieve, despite extensively experimenting with different hyperparameters. This again alludes to the algorithm's optimization instability.

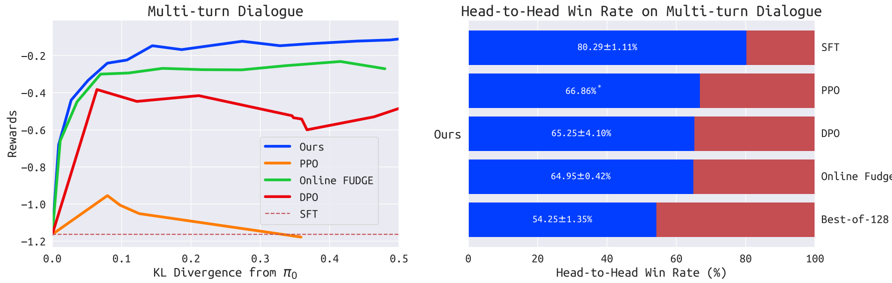

Figure 3: Performance on the Anthropic's HH dataset using LLaMA-2 7B model. VAS outperforms strong baselines like DPO and achieves comparable results to Best-of-128. We report the average win rate and standard deviation (error bar) across three random seeds.

3.3 Composing multiple alignment objectives

Task and Experimental Details We further extend the task of aligning to multiple sets of human preferences – specifically, improving the three axes of Attribution, Main Ideas, and Conciseness simultaneously. To transform VAS into a multi-reward optimization algorithm, we can take advantage of the fact that the Value is a linear function of the reward Sutton & Barto (2018). Therefore, VAS allows us to linearly combine the three Value estimators without any further re-training.

For baselines, we compare against Multi-objective PPO, a PPO training where the reward is the sum of the three separate reward functions. Moreover, we compare against Rewarded Soup Rame et al. (2023), an alternative approach to extend PPO for multi-alignment. Rewarded Soup trains a PPO model for each reward separately and averages the weights of these models during inference.

Results As seen in Figure 7, VAS effectively maximizes the rewards on each axis without sacrificing the performance in another and outperforms both Mo-PPO and Rewarded Soup. We observe an interesting phenomenon with Mo-PPO, where the different reward models compete with each other over the course of the training and the policy exhibits mode-switching behavior, alternating between optimizing for one reward model over the other. At one point, Mo-PPO (middle) is able to learn a reasonably good balance, but at the end of the training, Mo-PPO (last) collapses on one of the axes. This shows another example of the instabilities in PPO training.

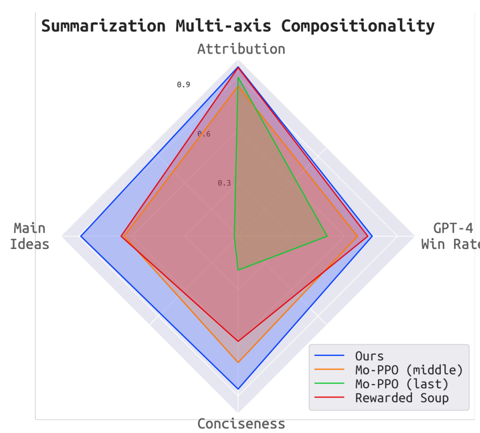

Figure 4: VAS is able to maximize multiple rewards simultaneously without sacrificing performance in any axis.

3.4 Improving GPT-3.5 API Tool-use

An important advantage of VAS is that it can work with black-box LLMs. We demonstrate this capability by teaching GPT-3.5 to use an API tool. Following (Xu et al., 2023), we experiment with the task of learning to use *Home Search* API. In this task, the model is provided an explanation of the APIs (zero-shot) and asked to perform a sequence of API calls to complete a search given the user query. To train VAS, we first collect 8K query examples and GPT-3.5 responses to these queries. Then, we trained a Pythia-1B model as a Value estimator using the success or fail signal as the reward. At inference, we query GPT-3.5 for the top-5 logits of the next token through OpenAI's API, augment them with our Value estimates, and sample the token with the highest combined score.

GPT-3.5 (zero-shot) only succeeds 17.4% of the time. Its performance improves to 27.9% with VAS. Adding an in-context example (one-shot) gets a performance of 62.8%. However, when VAS is combined with an in-context example, we improve the performance to 84.5%, further improving it by 22.3%. This demonstration sheds light on our approach as an *complementary* approach to teaching even closed-source, proprietary models how to use custom APIs.

REFERENCES

Nikolay Babakov, David Dale, Ilya Gusev, Irina Krotova, and Alexander Panchenko. Don't lose the message while paraphrasing: A study on content preserving style transfer. In Elisabeth Métais, Farid Meziane, Vijayan Sugumaran, Warren Manning, and Stephan Reiff-Marganiec (eds.), *Natural Language Processing and Information Systems*, pp. 47–61, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-35320-8.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 182–189. JMLR Workshop and Conference Proceedings, 2011.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

Elizabeth Clark, Shruti Rijhwani, Sebastian Gehrmann, Joshua Maynez, Roee Aharoni, Vitaly Nikolaev, Thibault Sellam, Aditya Siddhant, Dipanjan Das, and Ankur P Parikh. Seahorse: A multilingual, multifaceted dataset for summarization evaluation. *arXiv preprint arXiv:2305.13194*, 2023.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback, 2023.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. *arXiv preprint arXiv:2312.09244*, 2023.

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pp. 2021–2030. PMLR, 2019.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.

Google. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.

Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

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, 2021.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274, 2002.

Tomasz Korbak, Ethan Perez, and Christopher L Buckley. Rl with kl penalties is better viewed as bayesian inference. *arXiv preprint arXiv:2205.11275*, 2022.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 2023.

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. Don't throw away your value model! making ppo even better via value-guided monte-carlo tree search decoding. *arXiv e-prints*, pp. arXiv–2309, 2023.

Sidharth Mudgal, Jong Lee, Harish Ganapathy, YaGuang Li, Tao Wang, Yanping Huang, Zhifeng Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, et al. Controlled decoding from language models. *arXiv preprint arXiv:2310.17022*, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pp. 16828–16847. PMLR, 2022.

OpenAI. Gpt-4 technical report, 2024.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pp. 745–750, 2007.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search, 2023.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.

Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization. *arXiv preprint arXiv:2210.01241*, 2022.

Alexandre Rame, Guillaume Couairon, Mustafa Shukor, Corentin Dancette, Jean-Baptiste Gaya, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *arXiv preprint arXiv:2306.04488*, 2023.

Alexandre Ramé, Guillaume Couairon, Mustafa Shukor, Corentin Dancette, Jean-Baptiste Gaya, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards, 2023.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

Charlie Snell, Ilya Kostrikov, Yi Su, Mengjiao Yang, and Sergey Levine. Offline rl for natural language generation with implicit language q learning. *arXiv preprint arXiv:2206.11871*, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yunhao Tang, Rémi Munos, Mark Rowland, and Michal Valko. Va-learning as a more efficient alternative to q-learning. *arXiv preprint arXiv:2305.18161*, 2023.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. Recursively summarizing books with human feedback, 2021.

Tianhao Wu, Banghua Zhu, Ruoyu Zhang, Zhaojin Wen, Kannan Ramchandran, and Jiantao Jiao. Pairwise proximal policy optimization: Harnessing relative feedback for llm alignment. *arXiv preprint arXiv:2310.00212*, 2023.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models, 2023.

Kevin Yang and Dan Klein. Fudge: Controlled text generation with future discriminators. *arXiv preprint arXiv:2104.05218*, 2021.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

A PROOFS AND DERIVATIONS.

A.1 SOLUTION TO THE KL-REGULARIZED RL PROBLEM

A.2 SOLUTION TO THE KL-REGULARIZED RL PROBLEM

The following derivation is known as Reward-Weighted-Regression (Peters & Schaal, 2007) and is closely related to the Bayesian perspective on KL-regularized RL (Korbak et al., 2022). As a reminder, the problem we aim to solve is:

$$\max_{\pi} \mathbb{E}_{s_T \sim \pi}[r(s_T)] \quad s.t \quad D_{KL}(\pi || \pi_0) < \epsilon$$

Where we We will start by adding a constant to the first part of the objective, transforming it into:

$$\max_{\pi} \mathbb{E}_{s_T \sim \pi}[r(s_T)] - \mathbb{E}_{s_T \sim \pi_0}[r(s_T)] \quad s.t \quad D_{KL}(\pi || \pi_0) < \epsilon$$

Using the Performance Difference Lemma (Lemma 6.1 from (Kakade & Langford, 2002)) we can write:

$$\mathbb{E}_{s_T \sim \pi}[r(s_T)] - \mathbb{E}_{s_T \sim \pi_0}[r(s_T)] = \mathbb{E}_{s_T \sim \pi}[Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t)]$$

And so our optimization problem became:

$$\max_{\pi} \mathbb{E}_{s_T \sim \pi}[Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t)] \quad s.t \quad D_{KL}(\pi || \pi_0) < \epsilon$$

With the additional constraint that $\pi(\cdot|s_t)$ is a probability distribution for all t. The constraints should hold for every state, and in particular under the state distribution induced by π. Therefore, the corresponding Lagrangian is:

$$\mathcal{L}(\pi, \vec{\lambda}_1, \vec{\lambda}_2) =$$

$$\mathbb{E}_{s_T \sim \pi}\left[Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t) + \lambda_1(s_t)\left(D_{KL}(\pi||\pi_0) - \epsilon\right) + \lambda_2(s_t)\left(\sum_T \pi(x_{t+1}|s_t) - 1\right)\right]$$

Where $\vec{\lambda}_1, \vec{\lambda}_2$ is state-dependent lagrange multipliers. Notice that if we consider only the decision at a specific timestep τ, the expectation depends only on the policy's distribution on that specific timestep, for example the first term:

$$\mathbb{E}_{s_T \sim \pi}[Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t)] = \sum_{x_{t+1}} \pi(x_{t+1}|s_t)[Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t)]$$

Plugging it in and taking the derivative at specific timestep t with respect to the probability of taking a specific action x_{t+1}:

$$\frac{\partial}{\partial \pi(x_{t+1}|s_t)} \mathcal{L}(\pi, \vec{\lambda}_1, \vec{\lambda}_2) = Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t) + \lambda_1(s_t)\left(\log \frac{\pi(x_{t+1}|s_t)}{\pi_0(x_{t+1}|s_t)} + 1\right) + \lambda_2(s_t)$$

The corresponding fixed-point solution is:

$$\pi(x_{t+1}|s_t) = \pi_0(x_{t+1}|s_t) e^{\frac{Q^{\pi_0}(x_{t+1}|s_t) - V^{\pi_0}(s_t}{\lambda_1(s_t)}} e^{\frac{\lambda_2(s_t)+1}{\lambda_1(s_t)}}$$

We can simplify the expression by considering all the terms that are independent of the choice of x_t as part of the normalization constant, which leaves us with:

$$\pi(x_{t+1}|s_t) \propto \pi_0(x_{t+1}|s_t) e^{\frac{1}{\lambda_1(s_t)} Q^{\pi_0}(x_{t+1}|s_t)}$$

$\square$

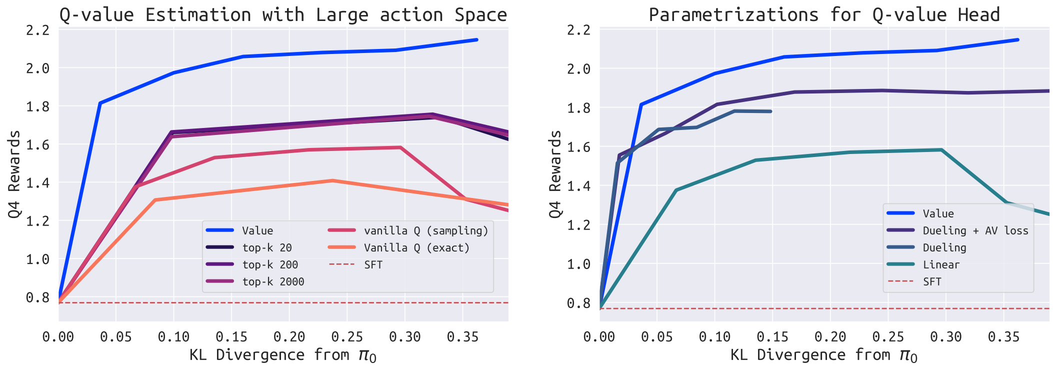

Figure 5:

B Q-Value estimation versus Value estimation.

B.1 Training Q Estimator for Text Generation

In our initial formulation, as presented in Equation 2, we propose augmenting the outputs of the baseline policy with the Q-values corresponding to each token. This step is helpful for enhancing the LLM's decoding process by considering the expected rewards associated with each potential token. Traditionally in reinforcement learning literature, Q-function estimators for discrete action spaces are parameterized as mappings from the entire state space to the action space. This typically involves a neural network architecture that processes the state as input, followed by a linear layer that outputs values spanning the action space dimension. The state space in text generation encompasses all conceivable text sequences, which is astronomically large. This is further compounded by the vast number of actions (tokens) typically involved, often numbering in the tens of thousands. Such a vast state and action space presents formidable optimization challenges.

Given the finite size of the training dataset, rare tokens (tokens with low probability under the original LLM) are scarcely represented, resulting in minimal updates to their corresponding weights. Consequently, these rare tokens may yield arbitrary values during inference, undermining the estimator's reliability. This is less of an issue for the state space since all input tokens share the same weights, and also, the backbone model is pretrained on a vast amount of data. To address and explore potential solutions to this challenge, we conducted a series of experiments. Through these investigations, we aim to provide insights into the complexities and practical challenges of applying Q-function estimators to text generation tasks, contributing to our decision to use a Value estimator instead.

B.2 Experiment Description

All the experiments described below were done on the SEAHORSE summarization dataset, specifically with property Q4 - attribution. However, our experiments indicate that all the conclusions also hold for the other two properties in this dataset. For the Q-value estimator, we use a Llama-2 7B backbone with a linear head on top of its last hidden layer. This linear head had an output dimension of 32k, which is the vocabulary size of the Llama-2 model family. The rest of the training details are exactly the same as described in section 3.1.

Our first experiment compared two ways to compute the target Q-value during Temporal Difference (TD) learning. In our experiment, we used TD(λ), which constructs this value as a linear combination of several n-step bootstrap targets and the final reward $\hat{v}(s_t) = \gamma^T \lambda^T r(s_T) + \sum_{i=t}^{T-1} \gamma^i \lambda^i [\gamma V^{\pi_0}(s_{i+1}) - V^{\pi_0}(s_i)]$. Since we don't have access to the true Values of the states, we need to estimate them using the Q function. The connection between Value and Q-values that come into hand here is $V^{\pi_0}(s_i) = \mathbb{E}_{x_i \sim \pi}[Q^{\pi_0}(s_{i-1}, x_i)]$. Having access to the output distribution of π_0, we can calculate this expression exactly or regard the tokens in the data as samples from this expectation. Figure X (left) shows the results of these two options as '**Vanilla Q (exact)**' and '**Vanilla Q (sampling)**'. The sampling method performs much better, and we hypothesize that this is because the sampled tokens are usually those with high probability. That way it disregards the rare tokens, which usually have bad Q estimators, in the bootstrapping process.

To investigate the effect of rare tokens on the performance further, we used the Q estimator trained with the sampling-based bootstrapping and, while decoding, evaluated only k tokens at each step. These **top-k** tokens were chosen as the ones with the highest probability under the base model. The results in Figure X (left) show that limiting the tokens to these with high probability helps a lot. We didn't find large performance gaps between different values of k up to two thousand. After that, the performance starts to degrade. This experiment supports the conclusion that with a linear head, it is hard to learn good Q-value estimation for all tokens.

After realizing the source of the poor performance of the Q estimator, we investigated with alternative parameterization to the Q-value estimator. Since learning Q-values for rare tokens is hard, we looked for a solution that will learn a value for them that will not have an effect on the decoding. **Dueling Networks** (Wang et al., 2016) does precisely that by decomposing the Q-value into a Value and Advantage. A dueling network has two heads, one to predict Values and one to predict Advantage, and during a forward pass it adds them together. The advantage head is initialized to zero, so rare tokens that do not change their prediction much will get an overall Q-value estimation which is just the Value. As can be seen from Figure X (right), this parameterization achieves significantly better results than the linear head. Therefore, to push the idea even further, we experimented with **Value Advantage Learning** (Tang et al., 2023), a recent learning objective that optimizes the Advantage head and the Value head separately. This represents the best results we were able to achieve using Q-value, and although they were significantly better than initial trials, they still didn't achieve the same performance as with the Value estimator.

B.3 ILQL

ILQL (Snell et al., 2022) is an offline RL algorithm designed for altering the behavior of LLMs given a fixed set of generated data (not necessarily from the initial LLM) annotated with a reward function. Unlike our algorithm, ILQL does not aim to learn a Q-value estimator for the current function but instead tries to learn a Q-value estimator for the optimal policy to the *unconstrained* RL problem. This is the policy that solely maximizes the reward, disregarding how far its output is from those of the initial policy π_0. After learning the Q-value estimator, ILQL uses during decoding a method similar to Equation 2. Although built for offline data, we experienced with an online version of ILQL to see if the algorithm implemented there could be useful for learning a Q-value estimator for text generation. The results in Figure X shows that ILQL can be used in the online setting, but is still underperformed compared to VAS.

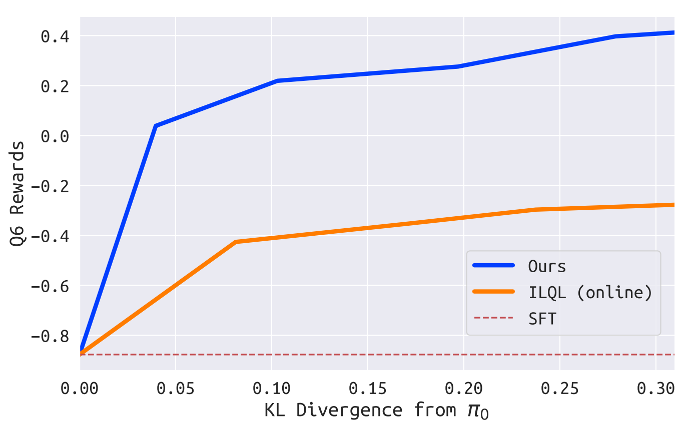

Figure 6:

C TRAINING DETAILS AND HYPERPARAMETERS.

We attempt to ensure that comparisons to baselines are fair. In particular, we take the following steps to ensure that. First, we perform a separate hyperparameter tuning for every algorithm. For every baseline, we use either the same number of hyperparameter combinations as for our method or more. In addition, when possible, we use the codebase released together with the original paper that proposes the baseline. In cases where this baseline is not suitable for our tasks, we use a well-known baseline to ensure proper implementation. The following sections explain the details for each one of the algorithms used in this paper.

C.1 VAS TRAINING

In all of our experiments, we parametrize the Value estimator as an LLM backbone with a linear head. For efficient training, we use LoRA parameterization (Hu et al., 2021) and train only the low-rank adapters and linear head. The data for training was collected using the SFT model with a sampling temperature of 0.7 and then labeled by the reward function. In addition, we also collected a validation dataset by first generating a dataset of half-completed responses and, for each one, generating ten completions and averaging their reward. The training objective we use is MSE between the Value estimator output to a Value target computed with TD(λ) (Sutton & Barto, 2018). For every task, we train three models with different learning rates and choose the one that achieves the lowest Value MSE on a validation set. For a full list of hyperparameters, see Table 1. During decoding, we evaluated a range of β values to report the KL-reward graphs. For GPT-4 evaluation, we choose the β value that achieves the highest reward.

	Summarization	Chat Dialogue	Tool Use	Personalization
Dataset	SEAHORSE	Anthropic HH	Custom Dataset	Custom Dataset
Reward Model	Trained based on T5-FLAN model	OpenAssistant/reward-model-deberta-v3-large-v2	Success Rate	Formality: s-nlp/roberta-base-formality-ranker Conciseness: Token Count
Value Model	LLaMA 7B	LLaMA-2 7B	Pythia 1B	LLaMA-2 7B
Lora Rank	60	32	32	32
Lora α	16	32	32	32
Batch Size	32	32	32	32
Learning Rate	1e-5, 5e-4, 1e-4	1e-4, 3e-4, 1e-3	1e-3	1e-4
Epochs	6	1	1	8
λ	0.95	0.95	0.98	0.95

Table 1: Hyperparameters for VAS training.

C.2 PPO TRAINING

For PPO training, we used the Alpaca Farm (Dubois et al., 2023) implementation. We parameterize both the model itself and the Value estimator with LoRA. The training data is collected through the run (as it is an on-policy algorithm) with a sampling temperature of 0.7. For the Summarization task, we have found that the KL coefficient has more effect on the performance than any other hyperparameter, and as such, did an ablation over three coefficient values. We report the results of the training (and checkpoint inside the training), which achieves the highest reward. For Multi-Objective PPO, we used the same training protocol, with the only difference being that we evaluated all three rewards during training and averaged their scores.

We found the training for the Chat Dialogue task to be unstable, resulting in poor performance. We tried two additional implementations of PPO [12] and tested over 20 different combinations of hyperparameters to no avail. As we mentioned before, this was reported by other research works as well (Rafailov et al., 2023). We report the results of the best run we were able to achieve.

For a full list of hyperparameters, see Table 2.

C.3 DPO TRAINING

For DPO training, we used the codebase released by its authors. We parameterize the model with LoRA with rank 32 and α 32. We conducted three trainings with different learning rates

[1]https://github.com/huggingface/trl

[2]https://github.com/CarperAI/trlx

	Summarization	Chat Dialogue
Dataset	SEAHORSE	Anthropic HH
Reward Model	Trained based on T5-FLAN model	OpenAssistant/reward-model-deberta-v3-large-v2
PolicyModel	LLaMA 7B	LLaMA-2 7B
Lora Rank	60	32
Lora Alpha	16	32
Rollout Batch Size	512	512
Step Batch Size	128	128
Learning Rate	1e-4	5e-6
KL coefficient	0.0067, 0.02, 0.05	0.05
Epochs	1	1
lambda	0.95	1
gamma	1	1

Table 2: Hyperparameters for PPO training.

$\{1e-4, 5e-5, 1e-5\}$ and reported the results of the model which achieved the highest margin on the validation set. The training was done with $\beta = 0.1$ and a batch size of 64. We evaluated the model every 10K training examples for both reward and win rate against SFT. We report the DPO result as the model that achieves the best win rate against SFT.

C.4 FUDGE Training

For the regular version of FUDGE, used in the Summarization task, we used Transformers[3] implementation of text classifier training and extended it for prefix classification. We parameterize the model with LoRA with rank 60 and α 16. We used a batch size of 32 and trained for three epochs until convergence. We performed three trainings with different learning rates $\{1e-3, 1e-4, 1e-5\}$ and reported the results for the model that achieved the highest accuracy on a validation set. For the Online FUDGE in the Chat Dialogue task, we used the same training protocol VAS and just changed the Value target to TD(1) to achieve a simple regression training.

C.5 Prompts for Training

C.5.1 SEAHORSE

For the task of summarization, we use the following prompt for every query:

```
Generate a one-sentence summary of this post.
```

C.5.2 Anthropic Helpfuless and Harmfulessness

For the task of multi-turn chat dialogue, we do not use any additional prompt and simply query the langauge model with the conversation between the human and assistant up until the assistant's response.

C.5.3 GPT-3.5 Tool-use

Following is the prompt containing brief description of the API functions that we use to query GPT-3.5:

```
I have the following set of API:

# To set home types for search. For home buying, home_types
choices are: "House", "Townhouse", "Condo", "Land",
"Multi-family", "Mobile", "Co-op"; for home renting, home_types
choices are: "House", "Townhouse", "Condo", "Apartment".
```

[3]https://github.com/huggingface/transformers

```
API.select_home_type(home_types: List[str])

# To specify whether to search homes for buying or renting.
'value' can be chosen from ['buy', 'rent']. This function must
be called after setting the location and before setting any other
criteria.
API.set_buy_or_rent(value: str)

# To set the minimum home size in square feet
API.set_min_square_feet(value: int)

# To set the maximum home size in square feet
API.set_max_square_feet(value: int)

# To set the floor number
API.set_floor_number(value: int)

# To set the maximum home price in dollars
API.set_max_price(value: int)

# To set the minimum home price in dollars
API.set_min_price(value: int)

# To set the location for the search area. This function must be
called before setting any criteria.
API.set_location(value: string)

# Submit criterion to get search results. This function should be
called after setting all the criterion.
API.search()

# To set the maximum commute time in minute to your office
location, assuming the office location is already defined
API.set_max_commute_time(value: int)

# To set the number of bathroom(s)
API.set_num_baths(value: float)

# To set the number of garage(s)
API.set_num_garages(value: int)

# To set the number of balconies
API.set_num_balconies(value: int)

# To set the number of bedroom(s)
API.set_num_beds(value: int)

# To set the number of swimming pool(s)
API.set_num_swimming_pools(value: int)

-------------

Task: [USER INPUT]
 Return only the relevant API calls, not any other kind of
response.
Action:
```

C.5.4 PERSONALIZATION

For the the task of personalization, we use the following prompt for every query:

```
    Please write a one paragraph explanation about the following
topic.
```

D RELATED WORK

Reinforcement Learning for Language Models. The most commonly used RL algorithm for tuning LLMs is Proximal Policy Optimization (PPO, (Schulman et al., 2017)) and its variants (Ramamurthy et al., 2022; Wu et al., 2023). Most of these are actor-critic algorithms (Sutton & Barto, 2018), which iterate between learning a Value estimator for the current policy and using this estimator to improve the policy. This bi-level process can lead to instabilities since errors, originating from a Value estimator that has not converged yet and the difficulties of constrained policy optimization, amplify each other (Fu et al., 2019; Van Hasselt et al., 2018). Moreover, training a Value function for an ever-changing policy creates optimization difficulties (Igl et al., 2020; Nikishin et al., 2022). Unlike these methods, our algorithm utilizes a closed-form solution to extract the optimal constrained policy given a Value function, bypassing the policy optimization step. In addition, the Value function is of the initial policy, which is fixed during the whole process. By that, we avoid all of the difficulties mentioned above.

Of most similarity to our work is ILQL (Snell et al., 2022), which proposes a Q-learning algorithm for steering LLMs. However, their work focuses on the offline RL setting, while ours assumes online interactions with the environment. We further discuss these similarities and differences in Appendix B.

Search-based algorithms for LLM decoding. There are several levels of granularity and different heuristics one can adopt for search. BoN performs search at the sequence level and uses a reward model as its search heuristic. Monte Carlo Tree Search Decoding (Liu et al., 2023), on the other hand, uses a Value function at the token-level to encourage sampling high-reward responses and then chooses the best sequence in the same manner as BoN. These algorithms and their variants have been shown to achieve remarkable performance but are expensive to compute (Nakano et al., 2021; Eisenstein et al., 2023).

Another notable line of work (Dathathri et al., 2019; Krause et al., 2020) introduces a family of search algorithms that performs a token-level search by using a classifier as the decoding heuristic. In each decoding step, they augment the LLM distribution with the classifier predictions and sample from it. The closest to our work among these algorithms is (Yang & Klein, 2021), which trains the classifier on partial responses from an offline dataset. In contrast to these prior methods, our work leverages the RL framework for training the search heuristic and casts the search objective as a constrained reward optimization.

Concurrently and independently of our work, Controlled Decoding (Mudgal et al., 2023) explores a similar method to use the Q function to guide the decoding process.

E ADDITIONAL RESULTS.

E.1 SUMMARIZATION HEAD-TO-HEAD WIN RATES

We further evaluate the performance of our models by directly comparing the generated responses from different models against each other with GPT-4 and present the results in Table 3. VAS clearly outperforms the baselines, with an average 60% win rate. More importantly, we approximately match the performance of Best-of-128.

E.2 MULTI-AXIS

Task In the real world, alignment to multiple different axes is required to model the complex set of human preferences. Thus, we further extend the evaluation of our algorithm's capabilities in solving

Model	Attribution	Main Ideas	Conciseness	Avg.
SFT	73.1%	77.0%	74.9%	75.0%
PPO	64.8%	59.3%	56.2%	60.1%
PPO + MCTS	62.2%	54.7%	54.0%	57.0%
FUDGE	59.2%	61.1%	58.8%	59.7%
BoN-128	49.7%	48.3%	49.2%	49.1%

Table 3: Head-to-head comparison between our algorithm to various baselines, as judged by GPT-4. Our model outperforms PPO and FUDGE and is almost on par with Best-of-128, while having lower inference compute cost.

multi-axis alignment. Specifically, we focus on the task of improving the three axes of Attribution, Main Ideas, and Conciseness simultaneously. Under the RL framework, multi-axis optimization is usually framed as maximizing a linear combination of different reward functions (Hayes et al., 2022). To transform VAS into a multi-reward optimization algorithm, we can take advantage of the fact that the Value is a linear function of the reward (Sutton & Barto, 2018). Therefore, maximizing a linear combination of rewards is equivalent to performing VAS with a linear combination of the respective Value estimators. It is important to mention that these estimators do not need to be jointly trained and are combined only during inference.

Experimental Details and Baselines We examine this idea by combining the Value estimator for all three axes with equal weighting. All other details are equivalent to those from the single-axis experiments. For baselines, we compare against **Multi-Objective PPO**, a PPO training where the reward is the sum of the three separate reward functions. Moreover, we compare against **Rewarded Soup** (Rame et al., 2023), an alternative approach to extend PPO for multi-alignment. Rewarded Soup trains a PPO model for each reward separately and averages the weights of these models during inference.

Results As seen in Figure 7, VAS effectively maximizes the rewards on each axis without sacrificing the performance in another and outperforms both MO-PPO and Rewarded Soup. We observe an interesting phenomenon with MO-PPO, where the different reward models compete with each other over the course of the training, and the policy exhibits mode-switching behavior, alternating between optimizing for one reward model over the other. At one point, as shown with MO-PPO (middle), it is able to learn a reasonably good balance, but at the end of the training, as seen with MO-PPO (last), the model collapses on one of the axes. This gives another example of the instabilities in PPO training.

E.3 Personalization

Task Unlike in the previous two sets of experiments, for some behaviors, it is not always desirable to simply maximize the rewards but control it dynamically. This is important in the problem of personalization, where users have different preferences for the style of responses. Prompt engineering is a solution, but it can be time-consuming and computationally expensive to optimize the prompts to get the exact degree of an attribute (Pryzant et al., 2023; Deng et al., 2022). In this experiment, we show that VAS is a complementary approach to prompting that allows easy tuning at inference time. This can be done by modulating the β parameter to control the weight between the original LLM and the Value augmentation. In contrast, control at inference-time is not possible with PPO, since a new policy needs to be trained for each different configuration. We show this fine-grained control capability over the axes of **Formality** and **Verbosity**.

Experimental Details For the initial policy, we fine-tune LLaMA-2-7B on Alpaca instruction following dataset (Taori et al., 2023) to get an SFT model that can follow human instructions. For Formality, we use a pre-trained reward model from (Babakov et al., 2023) based on Roberta Base (125M parameters) trained to predict whether a sentence is formal or not. For Verbosity, we use a numerical function of the sequence length as the response. We first collect 15,000 different topics

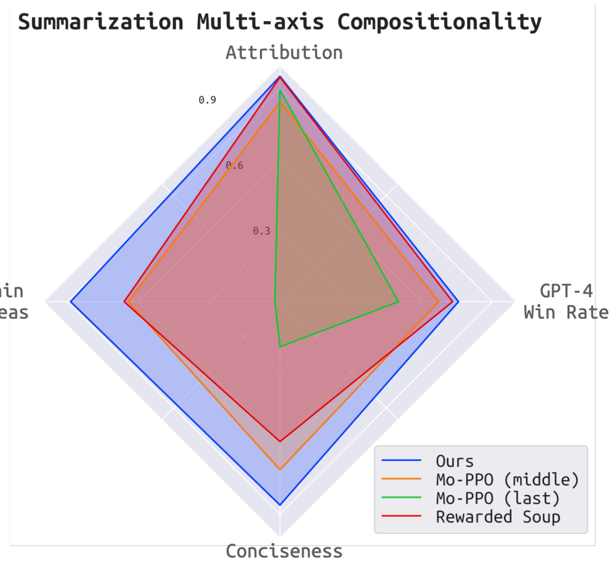

Figure 7: VAS is able to maximize multiple rewards simultaneously without sacrificing performance in any axis. Mode-switching behavior can be observed during the Mo-PPO training (see the comparison between middle and last), which once again alludes to its inherent instability in optimization.

(e.g., data compression, fruit wines, etc.) and use this data in tandem with the aforementioned reward models to train our Value estimators, which are also LLaMA-2-7B models.

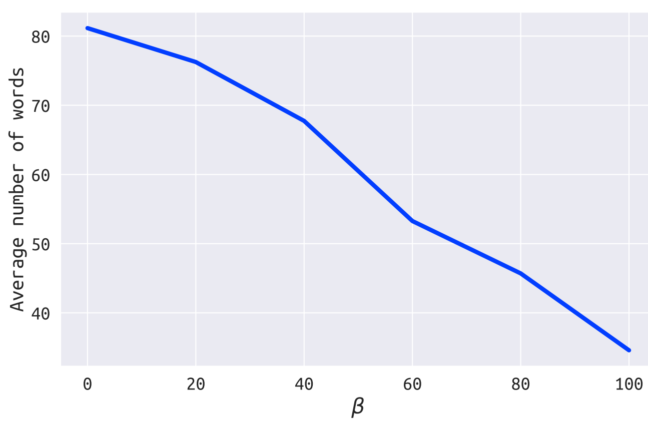

Figure 8: As we increase β during decoding, the average number of words per response decreases monotonically. This illustrates the ease of tuning LLMs' response lengths and versatility in reaching any point in the curve.

Results In Figure 8, we vary the β, the weight of our Value estimator, when augmenting the initial model. We clearly observe that there is a monotonic decrease in the average number of words per response as we increase β. This result illustrates that not only can we intuitively and easily change the behavior of the LLM, but also our algorithm has the versatility to reach any desired point in this alignment axis. For qualitative evaluation, we showcase how the response to the same prompt evolves when varying the β parameter for the axis of Formality. Figure 9 shows that the model starts to generate more causal responses when β is lower and more formal, technical responses when β is higher. Further examples can be found in Appendix D.

E.4 Learning Algorithms

VAS effectively leverages a Value function to guide the original LLM via token-level search, but how to best train a Value estimator and how to best guide the search (by directly augmenting the output

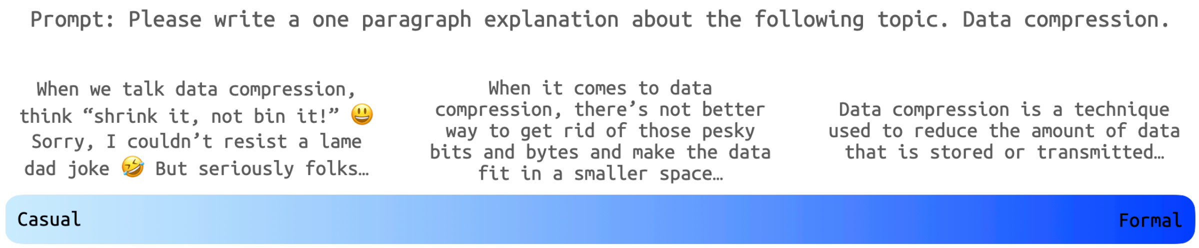

Figure 9: Qualitative results on Formality. By simply controlling β, our model easily varies from generating casual responses with emojis on the left to generating formal, technical answers on the right.

probabilities of the LLM) are open questions. To study this question, we performed an ablation study of our method. First, we removed the augmented Value given to the tokens not in the top-k. This design choice is a heuristic to minimize the effect of our approximation by using only top-k Values. As can be seen Figure 10, this causes quite a considerable degradation in performance.

In addition, we simplified the Value function training procedure and used TD(0) instead of TD(λ). This causes a small degradation in results, probably as a result of higher value estimation error.

E.5 Can we use small models to align bigger ones?

VAS proposes to align LLMs at inference-time by using the Value function as a search heuristic. Compared to policy-optimization methods, some of the computational costs move from training the policy to inference. This naturally raises the question of how our algorithm's computational cost compares against others. Let m be the compute (in FLOPS) needed by an LLM to process a single token, and n the compute needed by a secondary model (either Value estimator or reward function) to do the same. Policy optimization methods directly optimize the base policy and, therefore, only require $T \cdot m$ FLOPS to generate a response of length T. BoN requires generating N responses and evaluating all of them with the reward model, requiring a total of $N \cdot T(n + m)$ FLOPS. VAS requires one pass over the base policy and k token evaluation with our Value estimator, requiring a total of $T(m + kn)$. When $n << m$, our method can be significantly more efficient than BoN and approach the computational cost of using a single policy.

This analysis motivates the use of smaller models for the Value estimator. To investigate this possibility, we re-iterated the Chat Dialogue experiment, using a 1B model from the Pythia family (Biderman et al., 2023) to guide a Llama-2 7B model. As seen in Figure 11, the performance degrades with model size reduction, but the smaller model can still achieve a significant win rate of 68% over SFT. We conclude that VAS enables flexibility in the trade-off between compute and performance.

E.6 Is the Value estimator generalized beyond the training data?

VAS builds upon the idea of using a Value estimator to guide the search. As explained in section **??**, the Value estimator is trained on a dataset of prompts and responses. Even when trained on a specific dataset of prompts, an ideal adaptation method should generalize beyond it. To test VAS generalization capabilities, we checked the performance of our VAS model on MT-Bench (Zheng et al., 2024). MT-Bench is a comprehensive benchmark for LLM evaluation, covering many topics such as math, science, writing, and more. The SFT model got a score of 3.38 on this benchmark, the DPO model got a score of 3.91 and VAS got a score of 4.29. This shows that VAS indeed generalizes to this new dataset. Interestingly, VAS also seems to generalize better than DPO.

E.7 How does the Value estimator's accuracy affect alignment performance?

Equation 3 shows that leveraging our Value estimator as the search heuristic leads to the optimal solution to the KL-constrained RL problem. Since we don't have access to the true Value function and only estimate it, we question the relationship between our estimator's accuracy and the final performance. To study the Value estimator's accuracy, we generate a held-out validation set on which we generate ten different completions starting from a partial answer and compute the mean

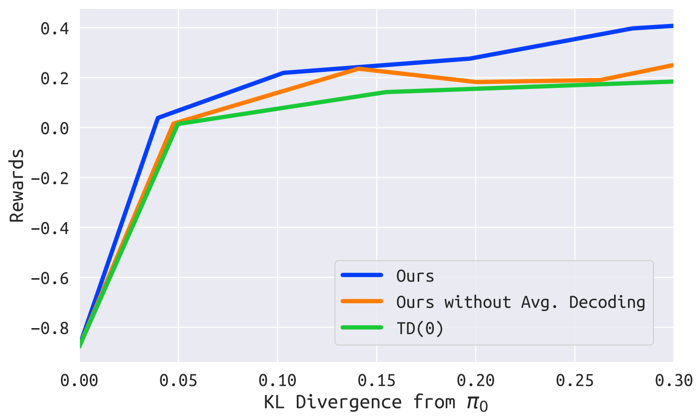

Figure 10: Abalation study on different design choices in our algorithm. The task is Summarization with Main Ideas as the reward function

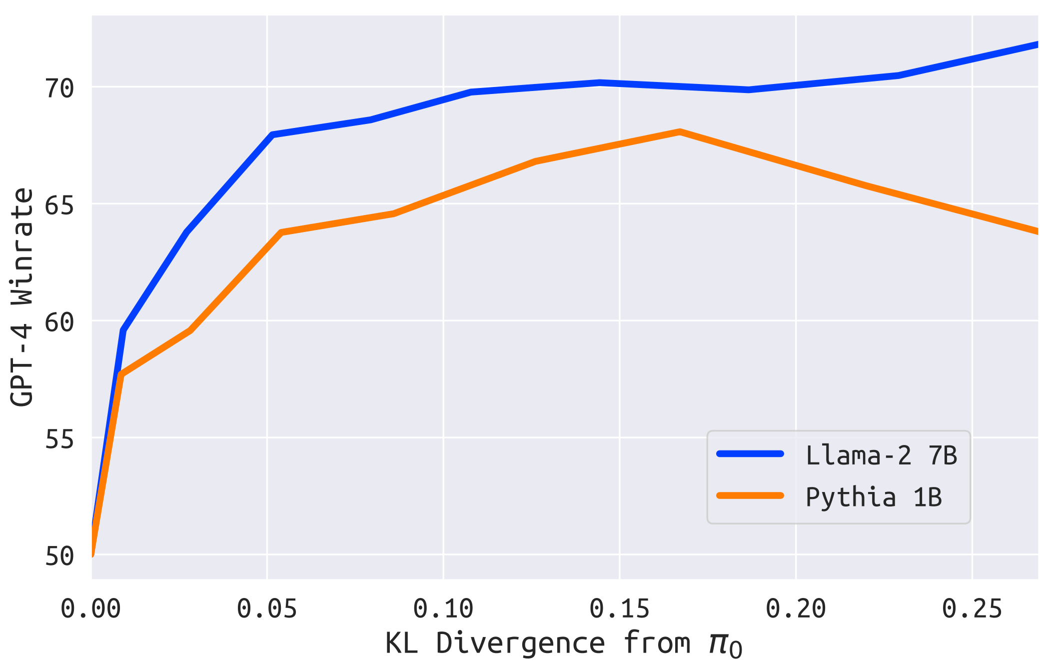

Figure 11: Comparison of different models size on the Chat Dialogue task. Smaller models achieve worse performance but require less computation during inference.

of the generations' rewards as the label. By varying the dataset size, we attain Value estimators of differing performances, and, as shown in Table 4, we observe a clear negative correlation clear negative correlation between the validation MSE and the rewards of the tuned LLM. In other words, as the Value estimator learns better estimates of the expected reward, the performance of the tuned model improves with respect to the reward model we are maximizing.

Dataset Size	Validation MSE ↓	Rewards ↑
32K	0.852	0.287
64K	0.811	0.355
96K	0.676	0.411

Table 4: We conduct an ablation along different dataset sizes to evaluate Value estimators of varying accuracies. These results illustrate that higher accuracy (lower validation MSE) is directly correlated with performance of the tuned LLM.

E.8 VARYING k

In Equation 4, we propose an approximation to the closed-form solution to the KL-regularized RL problem to realize it into a computationally feasible algorithm. As seen in the equation, k is a hyperparameter that directly influences the approximation and, to that end, we conduct an ablation study where we vary k and evaluate the performance. We conduct this ablation on the Conciseness task for summarization on the SEAHORSE dataset and present the results in Figure 12. Interestingly, for a fixed β of 5.0, lower k of 10 and 20 seem to be not only sufficient but more optimal than larger k of 100 and 200. We conjecture that this phenomenon may be related to the fact that lower k implicitly regularizes the tuned model to stay close to the initial model. However, further investigation is warranted to study how the optimal β changes with k.

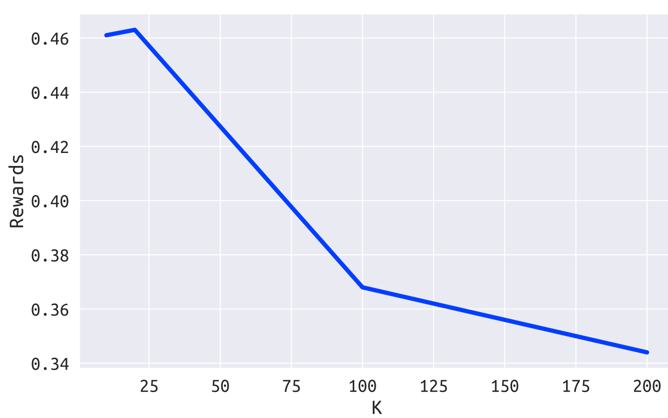

Figure 12: Varying K in VAS

F GPT-4 EVALUATION PROTOCOL

As a proxy of human judgment, we follow the GPT-4 evaluation protocol from AlpacaEval(Li et al., 2023) and evaluate the models' generated responses. We now describe the prompts that we use for the summarization and multi-turn chat dialogue tasks.

F.1 SUMMARIZATION PROMPT

```
    <|im_start|>system
You are helpful assistant whose goal is to simulate Diana's
preferred output for a given instruction.
```

```
Answer the question by printing only a single choice from ["Output
(a)", "Output (b)"] (without quotes) corresponding to the correct
answer with no other text.

Here’s some information about Diana:
- Diana loves literature and is highly cultivated
- Diana is a grammar freak
<|im_end|>
<|im_start|>user
## Annotation Guideline
In this task, we will ask you to select the preferred output AI
model’s responses to instructions.

You will read a batch of examples, which are composed of the
following:

1. an Instruction we give to the AI system
2. an Input that is provided along with the instruction
3. Output (a), the first output from the AI system
4. Output (b), the first output from the AI system

Your task is to decide which response is better for each example.
You should answer using only Output (a) or Output (b) depending on
which response is better.

## Annotation Example
To help you understand the annotation task, we provide a single
example below.
I will give an explanation for the correct answer, but you should
only answer with the preferred output.

### Example 1

#### Instruction 1:
Write a design document to describe a mobile game idea.

#### Input 1:
Game Idea: A battle royale game with a superhero and villain
theme.

#### Output (a) for example 1:
Design Document for a Mobile Game Idea:

1. Introduction:
A battle royale game with a superhero and villain theme.

2. Game Play:
The game is played on a large map with buildings, streets, and
structures spreading across the map. Players converge on the
center to compete in a winner-take-all showdown. The game includes
traditional battle royale elements such as scavenging, looting,
and trying to survive until the end. However, there is also a hero
system in place in which players can choose to take on the role
of a superhero or villain for the game. This adds some variety and
variety of tactics as players use their superpowers to wreak havoc
or use their villainsy to their own advantage.

3. Rewards:
```

```
As players compete in the battle royale game, they will receive
experience and level up their character. These levels reward
players with extra health, shields, and other all important
resources. Players can choose to customize their character in
various ways to make them stand out from the rest as well.

4. Levelling up:
At the end of each game, players will level up their character
and receive various bonuses depending on the level. These bonuses
could range from extra health to increased sprint speed or more
damage while in combat. As players gain levels, they can choose to
upgrade their character with more powerful abilities and powers.

5. Challenges:
Players can also complete

#### Output (b) for example 1:
The newly launched mobile game offers a thrilling battle royale
experience with a superhero and villain theme. Players jump out
of their cars and go head-to-head in a fight to the finish line.
Each round features a different set of powerful superheroes and
villainous villains, and the winner is the player who survives the
longest. Along the way, they can collect health items and upgrade
their powers to become the ultimate superhero or villain. The game
offers a lively and vibrant art style and a visually appealing
plot to keep the players engaged until the very end.

#### Which is best for example 1, Output (a) or Output (b)?:
Output (a)

Indeed, Output (a) is a design document as asked in the
instruction and gives more useful details than Output (b).

## Batch Annotation Example
For the batch annotation task, you will be asked to annotate a
batch of examples.
I.e. you will read all the examples first and then need to select
the preferred output.
Here’s an example of how you should be formatting your answers for
a batch of 3 examples.
Note that I will also give an explanation for the correct answer,
but you should only answer with the preferred output.

## Example 2

### Instruction 2:
Find the average age of the following people

### Input 2:
John (17), Laura (18), Robert (19), Edward (20).

### Output (a) for example 2:
The average age of the people in the given situation is 20 years
old.

### Output (b) for example 2:
The average age of the people listed is 19.5 years.

## Example 3
```

```
### Instruction 3:
Generate a to-do list with 5 tasks related to the following.

### Input 3:
keeping a neat room

### Output (a) for example 3:
Make beds
Organize closet
Shoot some hole in the walls
Dust lamps and furniture
Tidy drawers and shelves

### Output (b) for example 3:
chores:
1. Clean up after yourself and put clean clothes away.
2. Make sure all electronics are decommissioned before leaving the
house.
3. Remove all personal items and clutter from your room.
4. Vacuum and mop the floor.
5. Tidy up your desk and put all inboxes on hold.

## Example 4

### Instruction 4:
Rewrite the following sentence in a polite way

### Input 4:
Give it to me now.

### Output (a) for example 4:
Could you please give it to me, pending verification?

### Output (b) for example 4:
Can you please give me that item?

## Preferred Output for example 2-4:
Now give all the preferred outputs for the batch of 3 examples.

### Which is best for example 2, Output (a) or Output (b)?:
Output (b)

User explanation: Both outputs are wrong but Output (b) is
preferred. Indeed, the real answer is 18.5 years old, which is
closer to Output (b) than Output (a).

### Which is best for example 3, Output (a) or Output (b)?:
Output (b)

User explanation: Output (b) is better because Output (a) adds
to the to do list "Shoot some hole in the walls", which is not
related to tidying a room

### Which is best for example 4, Output (a) or Output (b)?:
Output (a)
```

```
User explanation: Output (a) is better because Output (b) adds
the sentence "pending verification", which is unnecessary and
unrelated to the instruction.

## Annotation starts below
Now is your turn. I will give you a batch of 5 examples.
You should read all the examples first and then select the
preferred answers by saying only Output (a) or Output (b) as
formatted above without explanation.

## Example 5

### Instruction 5:
{instruction}

### Input 5:
{input}

### Output (a) for example 5:
{output_1}

### Output (b) for example 5:
{output_2}

## Example 6

### Instruction 6:
{instruction}

### Input 6:
{input}

### Output (a) for example 6:
{output_1}

### Output (b) for example 6:
{output_2}

## Example 7

### Instruction 7:
{instruction}

### Input 7:
{input}

### Output (a) for example 7:
{output_1}

### Output (b) for example 7:
{output_2}

## Example 8

### Instruction 8:
{instruction}

### Input 8:
{input}
```

```
### Output (a) for example 8:
{output_1}

### Output (b) for example 8:
{output_2}

## Example 9

### Instruction 9:
{instruction}

### Input 9:
{input}

### Output (a) for example 9:
{output_1}

### Output (b) for example 9:
{output_2}

## Preferred Output for example 5-9:
Now give all the preferred outputs for the batch of 5 examples.
<|im_end|>
```

F.2 Multi-turn Chat Dialogue Prompt

```
    <|im_start|>system
You are helpful assistant whose goal is to decide on a preferred
answer to a question in a dialogue.
Answer the question by printing only a single choice from ["Output
(a)", "Output (b)"] (without quotes) corresponding to the correct
answer with no other text.
<|im_end|>
<|im_start|>user
In this task, we will ask you to select the preferred output AI
model’s responses.

You will read a batch of examples, which are composed of the
following:

1. A beginning of a conversation between an AI system
("Assistant") and a user ("User")
3. Output (a), the first output from the AI system
4. Output (b), the first output from the AI system

I will give you a batch of 5 examples. Your task is to decide
which response is better for each example. You should answer
using only Output (a) or Output (b) depending on which response
is better.

You should read all the examples first and then select the
preferred answers by saying only Output (a) or Output (b) as
formatted above without explanation.

## Example 1

### Instruction 1:
{instruction}
```

```
### Output (a) for example 1:
{output_1}

### Output (b) for example 1:
{output_2}

## Example 2

### Instruction 2:
{instruction}

### Output (a) for example 2:
{output_1}

### Output (b) for example 2:
{output_2}

## Example 3

### Instruction 3:
{instruction}

### Output (a) for example 3:
{output_1}

### Output (b) for example 3:
{output_2}

## Example 4

### Instruction 4:
{instruction}

### Output (a) for example 4:
{output_1}

### Output (b) for example 4:
{output_2}

## Example 5

### Instruction 5:
{instruction}

### Output (a) for example 5:
{output_1}

### Output (b) for example 5:
{output_2}

## Preferred Output for example 1-5:
Now give all the preferred outputs for the batch of 5 examples.
<|im_end|>
```