

Synthetic Data for any Differentiable Target

Tristan Thrush, Sung Min Park, Herman Brunborg, Luke Bailey,
Marcel Roed, Neil Band, Christopher Potts & Tatsunori Hashimoto
Stanford University
{tthrush, cgpotts, thashim}@stanford.edu

Abstract

What are the limits of controlling language models via synthetic training data? We develop a reinforcement learning (RL) primitive, the Dataset Policy Gradient (DPG), which can precisely optimize synthetic data generators to produce a dataset of targeted examples. When used for supervised fine-tuning (SFT) of a target model, these examples cause the target model to do well on a differentiable metric of our choice. Our approach achieves this by taking exact data attribution via higher-order gradients and using those scores as policy gradient rewards. We prove that this procedure closely approximates the true, intractable gradient for the synthetic data generator. To illustrate the potential of DPG, we show that, using only SFT on generated examples, we can cause the target model’s LM-head weights to (1) embed a QR code, (2) embed the pattern 67, and (3) have lower ℓ^2 norm. We additionally show that we can cause the generator to (4) rephrase inputs in a new language and (5) produce a specific UUID, even though neither of these objectives is conveyed in the generator’s input prompts. These findings suggest that DPG is a powerful and flexible technique for shaping model properties using only synthetic training examples. We release the code for our project at <https://github.com/TristanThrush/dataset-policy-gradients>.

1 Introduction

Synthetic training data has recently gained significant interest (Wang et al., 2023; Taori et al., 2023; Yang et al., 2025a; Ruan et al., 2025), but how finely can we control synthetic data generation? It is well-attested that training examples (real and synthetic) can communicate unexpected information to language models even in the context of simple supervised fine-tuning (SFT). Recent prominent examples include emergent misalignment (Betley et al., 2026; Chua et al., 2025), subliminal learning (Cloud et al., 2025; Betley et al., 2025), data poisoning from harmless inputs (Kong et al., 2025), and model provenance (Kuditipudi et al., 2025). Is there a way to tractably train a synthetic data generator that produces training data targeting any phenomena we choose?

Intuitively, straightforward reinforcement learning techniques could be used to optimize synthetic data generators directly for downstream metrics. Every time a dataset is generated by our policy, we could train a target model on it and measure a metric of interest from the model. We could then use this metric as a single reward for the entire dataset and perform a policy gradient step. However, this approach is computationally prohibitive because it provides only a single reward for a full run of inner target model training and evaluation.

In this work, we present the Dataset Policy Gradient (DPG), a principled RL approach that enables us to generate synthetic training data for any differentiable downstream target. With our method, rewards are at the level of individual synthetic texts, instead of the dataset level. This method opens the door to a wide range of applications in which training examples are chosen or synthesized with the goal of imbuing a target model with a specific property.

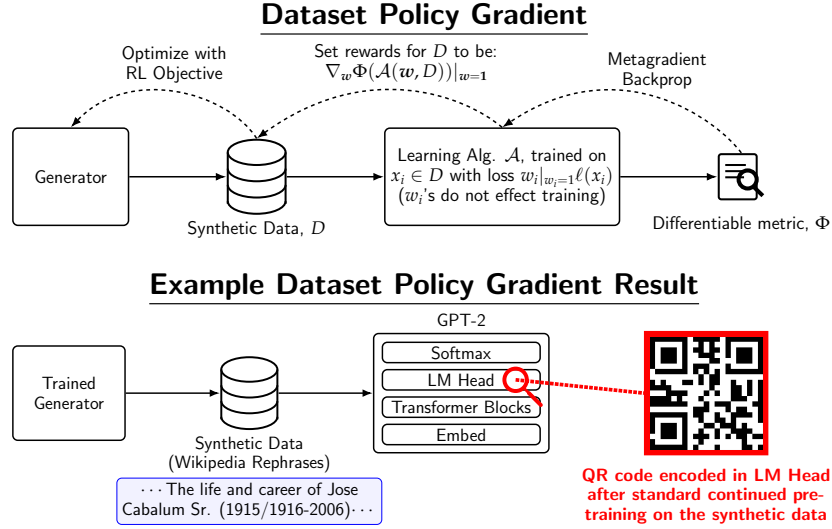


Figure 1: Dataset Policy Gradients allow us to generate synthetic training data for any differentiable target. For example, our generator can learn to generate special Wikipedia article rephrases. When used for continued pretraining of GPT-2, these rephrases turn the upper left 21x21 patch of GPT-2’s LM head weight matrix into the QR code seen here (when subtracted from the initial weights, sign’d, and visualized as a greyscale image). The text sample in this figure is the first item in the synthetic dataset, which we generated with a temperature of 1 (noisy data still produce the result).

Our approach leverages the meta-learning results of [Raghu et al. \(2021\)](#), and the recent improvements from [Engstrom et al. \(2025\)](#). These papers demonstrate how to compute metagradients (gradients of hyperparameters of the training process) tractably at the scale of LLM training. The metagradient enables backpropagation from a differentiable post-training metric (e.g., loss on a benchmark) to parameters of the training process (e.g., optimization hyperparameters such as learning rate schedules). Importantly, it is also tractable to compute metagradients for training example weights, if training occurs with a data-weighted loss. The memory and compute footprint is a roughly 3–5 $\times$ constant-factor increase from standard training. This leads to the key insight for our method: we can incorporate this metagradient-based data valuation approach into an RL procedure to generate targeted synthetic training data.

The DPG approach is a flexible framework. For the experiments in this paper, we use the configuration in Figure 1, top: a generator creates a pool of synthetic examples D , which are the inputs to learning algorithm $\mathcal{A}$. This learning algorithm trains a target LM on D with example-level training loss weights w_i set to 1. Then, the target LM is evaluated against a differentiable metric, Φ . The metagradient of Φ with respect to the w_i s determines a reward that is used to update the generator using Group Relative Policy Optimization (GRPO) ([Shao et al., 2024](#)). The trained generator produces examples that, if used to train a target LM with standard SFT, lead that LM to do well on Φ . In Section 3.2, we prove that the resulting policy gradient of this approach approximates the desired intractable policy gradient for the synthetic data generator, under standard assumptions.

We seek to test the limits of our method by experimenting with unusual choices of Φ . In our first experiments, we demonstrate that the generator produces examples that have a specific effect on the target model: encoding a QR code (Section 4.1) and the pattern 67 (Section 4.2) in the LM-head weights of the target model, and lowering the ℓ^2 norm of the LM-head weights (Section 4.3). We then directly assess the generator, showing that the Dataset Policy Gradient can guide it to rephrase Wikipedia articles in a new language (Section 4.4) and produce a specific UUID (Section 4.5), without any prompting for these behaviors.

In our experiments, we perform ablations to disentangle which aspects of the metagradient computation are essential in driving performance. For our QR code, 67, and ℓ^2 norm experiments, we find that computing metagradients with respect to several gradient descent steps of target model training is helpful. For the other experiments, we used a larger model as our target model and only tried one step of target model training for metagradient computation, due to compute constraints. We also find that the choice of target model optimizer (Adam vs. SGD) in the computation of the metagradient is a significant factor. Where we used SGD in learning algorithm $\mathcal{A}$ (Figure 1), the trained generator’s synthetic data typically did not cause the target model to perform well on Φ (even if Adam was used in after-the-fact training of the target model), whereas Adam is successful in algorithm $\mathcal{A}$. In the single-step case for SGD, the metagradient reduces to standard gradient-of-target and gradient-of-train dot-product approximations to influence functions (Koh & Liang, 2017). By contrast, where Adam is the optimizer, there are additional terms which make the metagradient different from approximations to typical influence functions, even in the single-step case.

Overall, our results provide evidence that the DPG framework allows for a new level of fine-grained control in synthetic training data generation, for the purpose of imbuing downstream models with specific properties – both desirable and undesirable.

2 Related Work

Synthetic data for language model training. Synthetic data is increasingly viewed as a key resource for language model performance gains (Wang et al., 2023; Taori et al., 2023; Maini et al., 2024; Abdin et al., 2024; Ruan et al., 2025; Yang et al., 2025b). Our contribution is orthogonal: instead of asking which synthetic data heuristics improve performance, we study how precisely synthetic data can be optimized – via metagradients – to induce targeted and even unconventional differentiable properties in trained models.

Training data attribution. We benefit from work attributing model behavior to individual training examples. Influence functions (Hampel, 1974; Koh & Liang, 2017; Bae et al., 2022) provide local estimates of how upweighting a training datum affects downstream performance. Recent work scales attribution ideas to modern LMs and multi-step training (Raghu et al., 2021; Ilyas et al., 2022; Park et al., 2023; Grosse et al., 2023; Xia et al., 2024; Thrush et al., 2025; Thudi et al., 2025; Engstrom et al., 2025; Calian et al., 2025). Data attribution is a subroutine in our work: we leverage the metagradients approach from Engstrom et al. (2025) to assign rewards to synthetic training examples generated by an RL policy.

Optimizing and editing training data. We focus on generating discrete synthetic training data from scratch. Other work has focused on targeted optimization of perturbations in differentiable training data, such as perturbing existing images (Such et al., 2019; Wang et al., 2020; Huang et al., 2021; Rosser et al., 2026). In the discrete data space, recent work includes RL approaches where models iteratively improve by generating synthetic training data for themselves, or through generating some other self-edit. In SEAL (Zweiger et al., 2025), LLMs generate candidate self-edits (directives on how to update their own weights); these directives are carried out, and edited LLMs are evaluated on downstream tasks. The performances of the edited LLMs are used directly as RL rewards, but this is intractable for our data generation tasks. MASS (Kaya & Rui, 2026) performs bilevel meta-adaptation using self-synthesized data at test time, computing a training data metagradient within an RL loop. MASS focuses on single datum adaptation at test time and computes the metagradient in the local one-train-step case without taking into account optimizer dynamics, analogous to an influence function approximation that lacks the more general metagradient critical for our tasks. In contrast to these methods, we prove that per-step metagradients provide accurate gradient signals that approximate the intractable full RL problem. Then, we optimize a policy that produces an entirely new training dataset targeting arbitrary differentiable training or post-training properties of an arbitrary target model over multiple training steps, taking into account arbitrary optimizers such as Adam (Kingma & Ba, 2015).

Optimizing inference data. Several approaches optimize prompts to elicit targeted behaviors at inference time (Pryzant et al., 2023; Khattab et al., 2024; Zou et al., 2023; Zhou et al., 2023; Opsahl-Ong et al., 2024; Agrawal et al., 2026). We instead optimize the generation of training data, so that learning itself induces desired behaviors.

3 Methods

We train a policy (i.e., the generator, π_θ) to generate training data for another model (i.e., the target model, trained in the RL loop within $\mathcal{A}$). The objective is to generate synthetic data D that increases the metric $\Phi(\mathcal{A}(D))$. Formally, we want to optimize π_θ via the objective

$$\max_{\pi_\theta} \mathbb{E}_{D \sim \pi_\theta} [\Phi(\mathcal{A}(D))],$$

but a direct approach is expensive: it involves a single RL reward over a dataset instead of a reward for each example in the dataset. In principle, the computational cost could be thousands or millions of times greater than a typical LM RL problem. Could we reduce this to a typical, per-example RL problem? Ideally, we want per-example rewards $r(x)$, for x in D , such that:

$$\nabla_\theta \mathbb{E}_{D \sim \pi_\theta} [\Phi(\mathcal{A}(D))] = \mathbb{E}_{D \sim \pi_\theta} \left[\sum_{x \in D} r(x) \nabla_\theta \log \pi_\theta(x) \right].$$

That is, taking a policy gradient step with respect to our per-example reward is equivalent to taking the intractable policy gradient step. This turns out to be possible and tractable. If $r(x)$ is defined as the exact influence of example x on the reward $\Phi(\mathcal{A}(D))$ through the training process, then the per-example policy gradient closely approximates the dataset-level policy gradient. In the next sections, we elaborate on how to take this exact influence (Section 3.1) and prove that this approximation is valid under natural assumptions (Section 3.2).

3.1 Algorithm

For our experiments, we use Group Relative Policy Optimization (GRPO) to train the generator (Shao et al., 2024), as shown in Algorithm 1. For every outer GRPO step, we can divide the set of policy generations into G training sets for a target model within the GRPO reward function. Optionally, we can also choose to do *cross-group batching*, combining all of these training sets into one training set, and running target model training once – this is more efficient. We run the inner target model training loop for potentially several optimization steps, with loss defined as

$$w_i \ell(\phi, x_i),$$

where ℓ is the standard causal language modeling loss, x_i is the i -th synthetic example, and w_i is the weight for the i -th example (with w set to 1 for target model training). Using the approach from Engstrom et al. (2025), we compute the gradient for these data weights:

$$\tau := \nabla_w \Phi(\mathcal{A}(w))|_{w=1},$$

A larger gradient for an example’s weight tells us that training on this example would improve the target metric more than training on an example with a smaller gradient. Motivated by this intuition, we use this gradient as the reward for our generator. In the following section, we provide a theoretical justification for this choice.

3.2 Theory

In our theory, we analyze a simplified variant of Algorithm 1 which replaces GRPO with the vanilla policy gradient update. We provide a finite-sample theorem where we optimize the target model with stochastic gradient descent (SGD). We use the metagradient computation method from Engstrom et al. (2025) to get $\tau_D := \nabla_w \Phi(\mathcal{A}(w, D))$, where $\mathcal{A}(w, D)$ is a learning algorithm that trains a target model on an n -sample dataset D with per-example weighted loss given by weights w . We generate D by sampling from a policy, and we use

Algorithm 1 An instance of the DPG framework, using GRPO (Online, Single-Turn). Note: $\mathcal{A}$ is a function – it is not stateful, so the target model trained in $\mathcal{A}$ resets after calling $\mathcal{A}$.

Require: Initial generator policy $\pi_{\theta_{\text{init}}}$; learning algorithm $\mathcal{A}$; differentiable metric Φ ; task prompts $\mathcal{P}$; hyperparameters M, G ; bool `use_cross_group_batching`.

Ensure: Trained policy π_{θ}

```

1:  $\pi_{\theta} \leftarrow \pi_{\theta_{\text{init}}}$ 
2: for  $\text{step} = 1, \dots, M$  do
3:   Sample a batch  $\mathcal{P}_b \sim \mathcal{P}$ 
4:   for  $q = 1, \dots, |\mathcal{P}_b|$  do
5:     Sample  $G$  outputs  $\{o_{g,q}\}_{g=1}^G \sim \pi_{\theta}(\cdot | q)$ 
6:   end for
7:   if use_cross_group_batching then
8:      $D \leftarrow \{o_{g,q}, \text{ for all } g \text{ and } q\}$  // Gather synthetic training dataset
9:      $\{r_{g,q}\} \leftarrow \nabla_w \Phi(\mathcal{A}(w, D))|_{w=1}$  // Call  $\mathcal{A}$ , compute metagradients, set rewards
10:  else
11:    for  $g = 1, \dots, G$  in parallel do
12:       $D_g \leftarrow \{o_{g,q}\}_{q=1}^{|\mathcal{P}_b|}$  // Gather synthetic training dataset
13:       $\{r_{g,q}\}_{q=1}^{|\mathcal{P}_b|} \leftarrow \nabla_w \Phi(\mathcal{A}(w, D_g))|_{w=1}$  // Call  $\mathcal{A}$ , compute metagradients, set rewards
14:    end for
15:  end if
16:  Compute group-relative advantages  $\hat{A}_{g,q}$ 
17:  Update  $\pi_{\theta}$  via the GRPO objective (Eq. 21 in Shao et al. (2024))
18: end for
19: return  $\pi_{\theta}$ 

```

our metagradient as the reward signal. Treating the τ_i as per-example rewards, we take the policy gradient step given by $\mathcal{G} = \tau_i \nabla_{\theta} \log \pi_{\theta}(x_i)$.

Now, let $F(\theta) := \mathbb{E}_{D \sim \pi_{\theta}}[\Phi(\mathcal{A}(D))]$. F is the target performance of a model trained on samples from π_{θ} . Taking gradient steps on F directly optimizes for our target, but this does not give us example-level rewards and it is not tractable in any of our experiments.

Now, let $F'(\theta, p) := \mathbb{E}_{D \sim p}[\Phi(\mathcal{A}(\pi_{\theta}/p, D))]$. Note that F' is the surrogate that we actually optimize in our DPG setup. Setting $w_i(\theta) = \frac{\pi_{\theta}(x_i)}{p(x_i)}$ and using the chain rule, we have:

$$\nabla_{\theta} F'(\theta, p) = \mathbb{E}_{D \sim p} \left[\sum_{i=1}^n \frac{\partial}{\partial w_i} \Phi(\mathcal{A}(w, D)) \frac{\pi_{\theta}(x_i)}{p(x_i)} \nabla_{\theta} \log \pi_{\theta}(x_i) \right]$$

Setting $\pi_{\theta} = p = \pi_{\theta_0}$, we see the metagradient update $\mathcal{G}$ is an unbiased stochastic gradient for F' . Via the following theorem, $\nabla_{\theta} F'$ accurately approximates the ideal gradient: $\nabla_{\theta} F$.

Theorem 3.1. Suppose we train the target model in $\mathcal{A}$ for T steps of minibatch stochastic gradient descent (SGD) with batch size B and a learning rate of η . Under suitable regularity conditions on smoothness (Appendix A, A1-A8), we have:

$$\sup_{\theta_0} \|\nabla_{\theta} F(\theta_0) - \nabla_{\theta} F'(\theta_0, \pi_{\theta_0})\| = O(\eta^{\frac{1}{4}} B^{-\frac{1}{2}} + \sqrt{\eta T})$$

N.B. – although it may be clear to some, the notation can be tricky to keep straight. In this equation, we take the gradient of F' with respect to only the first argument, evaluated at θ_0 , with p set to π_{θ_0} .

See Appendix A for a proof. This theorem shows that, under the first- and second-order smoothness assumptions listed in Appendix A, our metagradient reward policy gradient can approximate the desired policy gradient for the generator if $\mathcal{A}$ has the following properties: the batch size is large, and the step size is small relative to the number of training steps.

It is important to note that, even though our finite-sample theorem above assumes SGD, we find experimentally that it is essential to use Adam (Kingma & Ba, 2015) to train the target model in the computation of the metagradient. This remains true even when we use only a single step of target model training to compute the metagradient. We conjecture

that using Adam, like SGD, would also result in a reasonable finite-sample bound via $\sup_{\theta_0} \|\nabla_{\theta} F(\theta_0) - \nabla_{\theta} F'(\theta_0, \pi_{\theta_0})\|$, but still with some error: like SGD’s behavior, Adam’s behavior depends on the second moment of the target model’s loss gradient (which is different between F and F').

4 Results

We present experiments where we train synthetic data generators to target various metrics downstream of training a target model. Our target metrics are intentionally unusual: although we show steps toward practical applications such as benchmark targeting and data poisoning, the goal of the experiments is primarily to establish that DPG can optimize synthetic training data for arbitrary differentiable targets. We chose our experiment settings to showcase the generality of our method.

We first validate our pipeline end-to-end, generating synthetic train data that can precisely manipulate the weights of target models. We then analyze the generator’s output to determine whether the synthetic data are interpretable. In all of our experiments, the generator is initialized from Llama 3.2 Instruct (Grattafiori et al., 2024) and given Wikipedia¹ articles to paraphrase (prompt in Appendix G). It then learns through Dataset Policy Gradients, optimizing its paraphrases D to target a differentiable metric Φ of a learning algorithm $\mathcal{A}(D)$. The target model in $\mathcal{A}$ is initialized from Llama 3.2 Instruct as well, or GPT-2 (Radford et al., 2019), depending on the experiment. GPT-2 is used in experiments with several $\mathcal{A}$ training steps, because our compute constraints required us to use a smaller model.

All experiments use the instance of the DPG framework with GRPO and cross-group batching (see Appendix Figure 7), unless stated otherwise. The naive baseline of using Φ directly as the reward never uses cross-group batching (to get more reward signal) and also treats every example as coming from the same group for computing advantages (otherwise, the advantage calculation would render the rewards useless). All validations use Wikipedia articles not seen during training, unless stated otherwise. Hyperparameters for all experiments are in Appendix E.

We explored training the target model with both Adam and SGD for metagradient computation. For SGD, we tried up to 14 learning rates (LRs) starting at 1e-6, and increasing by factors of 2, until we found the optimal LR against final validation loss for each task. We did the same tuning for the naive approach (which uses Adam to train the target model but does not compute metagradients), and other baselines. There was no need to tune the LR for the metagradients + Adam approach.

Wherever we trained our generator via SGD in $\mathcal{A}$, we also used SGD in target model training to get validation results. The one exception is in Appendix C, where we trained a generator using SGD in $\mathcal{A}$, but validated its synthetic data by training a target model with Adam.

4.1 Encoding a QR Code in a Target Model’s LM Head

In this section, we ask: can we automatically craft synthetic data so precisely that it can embed a QR code into the weights of a model that trains on it? We make our target loss

$$\text{mean} \left(\ln \left(1 + e^{-sY \odot (P_c - P_i)} \right) \right),$$

where Y is a matrix of -1 ’s and 1 ’s representing the pattern that we want to encode into the target model, P_c is a chosen patch of the target model’s LM-head weight matrix in $\mathcal{A}$ after training, P_i is the same patch of the LM head before any synthetic training, and s is a hyperparameter that we set to 20 for all experiments. After target model training, we decode our image to see if it matches Y by taking the following expression: $\text{sign}(P_c - P_i)$.

For the QR code experiment, we set Y to be an arbitrarily chosen 21x21 QR code, and set our target model to be GPT-2. In each of the $M = 200$ GRPO steps, we do 96 steps of continued

¹Accessed in 2025 via <https://huggingface.co/datasets/wikimedia/wikipedia>

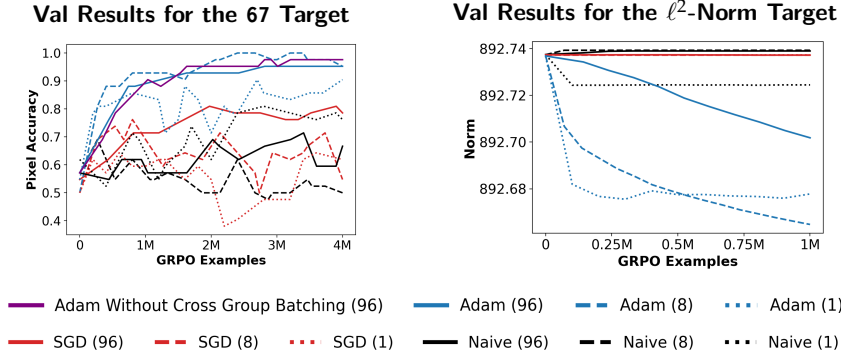


Figure 2: Here, we initialize the target model in $\mathcal{A}$ to be GPT-2, and explore exotic target metrics: the goal of the first metric is to encode the greyscale image 67 in the upper 6x7 patch of the sign’d LM-head weight updates to the target model. This number was chosen arbitrarily. The goal of the second metric is to lower the ℓ^2 norm of the target model’s LM head. The plots show validation performance as the GRPO process trains the generator. All validations are done with 96 steps of continued training on GPT-2. The (96), (8), and (1) notation denotes whether the generator was trained via metagradients with respect to an $\mathcal{A}$ that used 96, 8, or 1 step(s). We observe a weak correlation between $\mathcal{A}$ steps and validation performance, and generally more validation stability with more $\mathcal{A}$ steps.

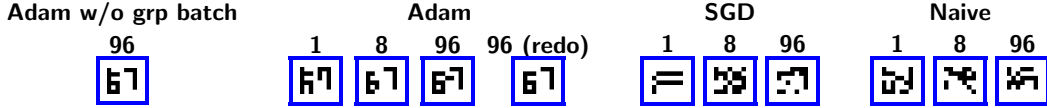


Figure 3: Final validation results for the 6x7 pixel images in the target models’ sign’d LM-head updates, after the generator was fully trained. The numbers above the images denote the number of target model training steps in $\mathcal{A}$ for metagradient computation. All validations were done with 96 target model training steps, using the corresponding optimizer; the difference is whether the generator was trained using a reward function with fewer $\mathcal{A}$ training steps. Only Adam with 96 steps in $\mathcal{A}$ for metagrads achieved a generator that got a perfect result (we were close with the initial 96 run, so we trained the generator again with a different random sample of Wikipedia prompts – we then got a perfect score).

pretraining on GPT-2 and then compute metagradients. We target the upper left 21x21 patch of GPT-2’s LM head. For each target model training step, we use a batch size of $B = 1024$ synthetic examples, so the synthetic data generator produces $96 \times 1024 = 98304$ Wikipedia rephrases per GRPO step. Due to compute constraints, we ran this experiment only once and did not conduct separate validations with unseen prompts during generator training. The final trained generator’s synthetic data, when used for 96 steps of continued pretraining on GPT-2, yields a $\text{sign}(P_c - \hat{P}_i)$ that is a scannable QR code (Figure 1).

4.2 Encoding 67 in a Target Model’s LM Head

Now, we investigate which elements of our DPG framework are essential for embedding images in model weights. We explore the use of SGD instead of Adam inside of $\mathcal{A}$. We also ask if it would be sufficient to train a target model in $\mathcal{A}$ with only 8 optimizer steps, or even 1 step, during generator training; what would happen if we ran a validation at the end of this generator’s training by training a target model on 96 steps – would we lose some performance? Here we present an array of experiments using the same setup as in Section 4.1, but in a scaled-down setting, where we set Y to be a 6x7-pixel image of the arbitrarily-chosen number 67. This enables us to run more experiments.

We set P_i to be the upper left 6x7 patch of GPT-2’s initial pretrained LM-head weight matrix. We set P_c to be the same LM-head weight patch after synthetic training. We run experiments

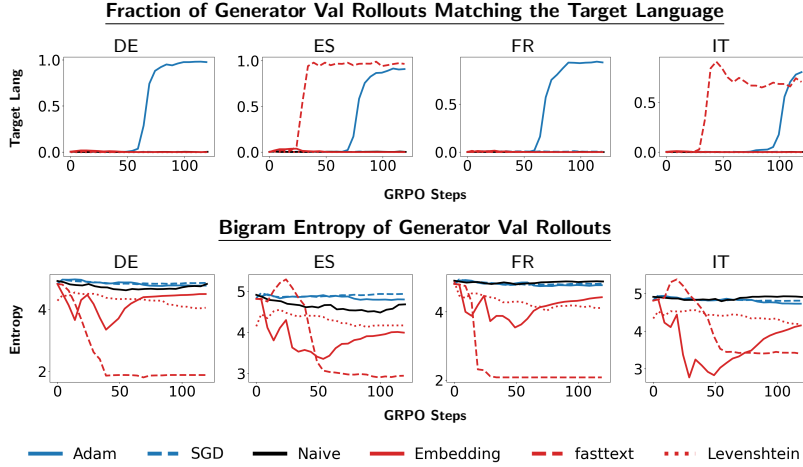


Figure 4: Generator results when setting Φ to be post-training loss on four multilingual LAMBADA (Paperno et al., 2016) translations from Black (2023): DE, ES, FR, and IT. We initialized the generator from Llama 3.2 Instruct. We initialized the target model in $\mathcal{A}$ also from Llama 3.2 Instruct. In each GRPO step, we conduct a single step of target model continued pretraining on the synthetic data before computing metagradients. When using Adam in $\mathcal{A}$, the generator learns the correct language, as judged by GPT 4.1 Nano (OpenAI, 2025). Baselines do not learn the correct language except in rare cases where their entropy quickly collapses and they repeatedly produce only a few words.

with 96 steps, 8 steps, and 1 step for computing metagradient rewards from $\mathcal{A}$, both with Adam and with SGD. We always validate using 96 steps of training on generated data. In the 96 step metagradient case, we use $M = 40$ GRPO steps with target model train batches of $B = 1024$ examples each. To keep the total number of examples the same in the 1 step and 8 step cases for GRPO generator training, we set $M = 3840$ and $M = 480$, respectively.

Validation progress in terms of the percent of pixels correct in $\text{sign}(P_c - P_i)$ is shown in Figure 2, and final validation images are seen in Figure 3. Using Adam in $\mathcal{A}$ with multiple target model training steps does best. The purple line shows the results when we use DPG GRPO without cross-group batching as seen in Appendix Figure 6 (the group size is 4, so the target model train batches for metagradients are 256 instead of the usual 1024). Examples of generator rollouts for trained generators are shown in Appendix D – we see that after using our DPG GRPO, the generations still appear to be normal Wikipedia rephrases, while sneakily encoding a 67 in a target model’s weights when used as training data.

4.3 Adjusting the Norm of Target Model Weights

In this section, we follow the same setup as in the previous section, changing only the target metric. We ask whether we can globally affect an entire weight matrix in a target model. So, the target metric here is to lower the ℓ^2 norm of GPT-2’s LM-head weight matrix.

Figure 2 shows the results of this experiment. We explored changing the optimizer and number of steps in $\mathcal{A}$ for metagrads, just as in Section 4.2, but we used $4\times$ fewer GRPO steps (M) in every case. Using DPG with Adam in $\mathcal{A}$, again, is the only approach to show reasonable progress. We also see that when GRPO training has an $\mathcal{A}$ with only a single training step, the performance plateaus early. This supports the idea that computing metagradients over several steps of synthetic pretraining can be useful. Note that, because the validation results are all for 96 steps of continued pretraining, the change in the ℓ^2 norm is capped – although the final ℓ^2 norm differences are small everywhere, the change may not remain small for larger amounts of CPT data.

4.4 Interpreting the Generated Data when Targeting Multilingual Validation Loss

We now switch our analysis from the target model to the trained generator: does it learn interpretable generations? It is hard to know what data it should generate to lower the target model’s norm or draw images in its weights. However, we would expect that if we made the target metric to lower the language modeling loss of the target model on a non-English language, the generator would eventually learn to rephrase the Wikipedia articles into that language. Is our DPG approach powerful enough to guide the generator to perform this translation, even if the prompt does not mention translation and the Wikipedia articles are all English? We find that the Adam version of our approach is able to teach the generator to accomplish this feat, while other baselines are not.

We conduct experiments in four different settings where Φ is language modeling loss on the train sets of the DE, ES, FR, and IT LAMBADA (Paperno et al., 2016) translations from Black (2023). We chose this benchmark for two reasons. First, it provides a setting where it is easy to determine whether the generator has learned a desirable behavior (translation into the target language in a storylike writing style). Second, we believe it is particularly challenging in our setup: the generator only receives English Wikipedia inputs, the prompts never request translation, and the reward signal is derived solely from downstream target model performance on the non-English benchmark. The generator must discover a nontrivial transformation solely through metagradient rewards on the English Wikipedia rephrases.

Note that the standard LAMBADA dataset only provides a single group of 5.15K examples, so we split it into train, val, and test sets of 2.32K, 515, and 2.32K examples, respectively. We only use the train set in our target metric. These splits were useful for our experiments in Appendix C, which we discuss later in this section. We used Llama 3.2 Instruct as the target model, and used only one target model training step both in $\mathcal{A}$ and for validation. Otherwise, the setup is the same as the previous experiments. We train the generator with $M = 120$ GRPO steps, using batches of $B = 1024$ synthetic data examples.

We implement a variety of new baselines for this section: “Embedding”, “fasttext”, and “Levenshtein”. The Embedding baseline computes average embedding similarity of each rollout example with the LAMBADA examples, and this is used as the reward for RL instead of metagradient weights. The embeddings used are from Aarsen (2025), and we use their provided similarity function. The fasttext baseline computes the fasttext language classification probability of the target language, for each rollout example, and uses this as the reward. The fasttext model we use is from Grave et al. (2018). Finally, the Levenshtein baseline uses as rewards the average negative Levenshtein distance (Levenshtein, 1966) between each rollout example and the LAMBADA examples.

We show in Figure 4 that the Adam version of DPG GRPO is the only algorithm to reliably teach the generator to translate its rephrases into the correct non-English language. The generator does this while maintaining the entropy of the rephrases (no clear mode collapses).

Appendix C shows that we can take Llama 3.2 Instruct (and Llama 3.2 Base, for which the generator was not explicitly optimized) and train it on 10M tokens from our tuned generator to get high benchmark performance relative to a variety of baselines. This amount of synthetic CPT data is more than the single step of training data for which the generator was explicitly optimized. In these validations, we train in PyTorch (Ansel et al., 2024), whereas the Llama 3.2 Instruct in $\mathcal{A}$ used a JAX (Bradbury et al., 2018) implementation. We also evaluate benchmark performance via perplexity in the Eleuther Eval Harness (Gao et al., 2024), which is slightly different than Φ ’s language modeling loss – yet there is transfer.

4.5 Interpreting the Generated Data when Targeting Loss on a UUID

If we set the target metric to be language modeling loss on another language, the generator will learn to produce its Wikipedia paraphrases in that language. But, just how powerful is the metagradient signal on the rephrases? Can we teach the generator to generate an unnatural 32-character UUID that appears nowhere in the initial generator rollouts?

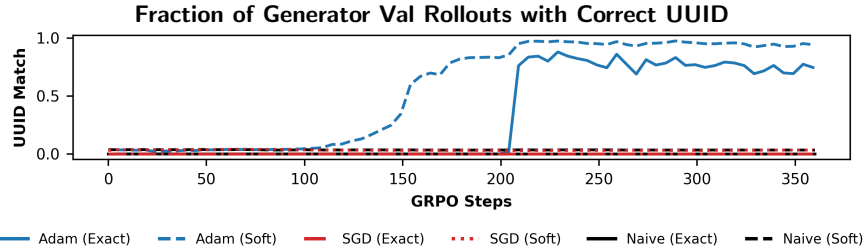


Figure 5: We keep the same setup as the LAMBADA cases, with the exception of changing Φ to be the target model’s post-training LM loss on a 32-character UUID. In this plot, we show two validation metrics: *Exact* requires the complete UUID to be in a rollout, and *Soft* finds the longest substring of the UUID in the rollout and gives points proportional to the fraction of the UUID present.

Here, we keep the same setting as the LAMBADA experiments, except: we change the target metric of the model from $\mathcal{A}$ to be language modeling loss on a 32-character UUID, conduct GRPO training for $3\times$ as long, and set generator validation sampling temperature to zero. The generator learns to produce the UUID in the Adam case. In the SGD and Naive cases, the generator never learns to generate any component of the UUID with higher frequency.

5 Conclusion

We introduced the Dataset Policy Gradient, a new RL primitive for generating synthetic training data that can be optimized for any differentiable training or post-training target metric. DPG is tractable: the computation of metagradients requires a roughly $3\text{--}5\times$ constant factor increase in memory and compute from target model training, and yields dense rewards. We also presented theoretical arguments that DPG RL keeps the policy gradient close to the ideal policy gradient, under typical assumptions. We then showcased that synthetic training data generated using DPG RL can draw images in LM weights, alter the ℓ^2 norm of LM weights, and target LM benchmarks, all through standard SFT. Interestingly, it was important to use Adam inside of $\mathcal{A}$ for the computation of metagradients. This suggests that it could be useful to revisit influence function results (Koh & Liang, 2017), which typically ignore the optimizer and the learning trajectory. Overall, this new framework for optimizing synthetic training data allows us to reach a new level of fine-grained targeting.

Implications

DPG may enable practitioners to intentionally steer models toward desirable capabilities using synthetic SFT examples. At the same time, this level of control has potential risks. If synthetic data generation can be optimized to induce arbitrary differentiable properties in trained models, adversaries could potentially craft subtle data poisoning attacks that target specific biases or behaviors. Understanding both the capabilities and risks of targeted synthetic data generation will be important as synthetic data becomes an increasingly central component of modern machine learning pipelines.

Acknowledgments

We thank Anupama Sridhar for conversations on the mathematical aspects of this work and feedback on our draft. We thank Christopher Mohri for conversations on the mathematical aspects of this work. TT is supported in part by the Stanford Graduate Fellowship and in part by the Amazon AI Fellowship. SP was supported in part by a HAI Hoffman-Yee grant. HB thanks the Aker Scholarship Foundation for financial support. LB is supported in part by the Stanford Graduate Fellowship and in part by the FLI Vitalik Buterin Fellowship. NB

acknowledges support from an NSF Graduate Research Fellowship, Quad Fellowship, and Mercor Graduate Fellowship. CP acknowledges support from Google and Open Philanthropy (Coefficient Giving). TH was supported by a grant by HAI, DSO labs, gifts from Open Philanthropy, Amazon, Schmidt Sciences, the Tianqiao and Chrissy Chen Foundation and a grant under the NSF CAREER IIS-2338866, ONR N00014-24-1-2609, and DARPA Cooperative Agreement HR00112520013. This work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred.

References

- Tom Aarsen. Train 400x faster static embedding models with sentence transformers. 2025. URL <https://huggingface.co/sentence-transformers/static-similarity-mrl-multilingual-v1>.
- 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, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report. *arXiv*. 2024.
- Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziemis, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *ICLR*. 2026.
- Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhersch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2024.
- Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*. 2022.
- Jan Betley, Jorio Cocola, Dylan Feng, James Chua, Andy Ardit, Anna Sztyber-Betley, and Owain Evans. Weird generalization and inductive backdoors: New ways to corrupt LLMs. *arXiv*. 2025.
- Jan Betley, Niels Warncke, Anna Sztyber-Betley, Daniel Tan, Xuchan Bao, Martín Soto, Megha Srivastava, Nathan Labenz, and Owain Evans. Training large language models on narrow tasks can lead to broad misalignment. *Nature*. 2026.
- Sid Black. Multilingual LAMBADA. 2023. URL https://huggingface.co/datasets/EleutherAI/lambada_openai.
- James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. 2018. URL <http://github.com/jax-ml/jax>.
- Dan A. Calian, Gregory Farquhar, Iurii Kemaev, Luisa M. Zintgraf, Matteo Hessel, Jeremy Shar, Junhyuk Oh, András György, Tom Schaul, Jeffrey Dean, Hado van Hasselt, and David Silver. DataRater: Meta-learned dataset curation. *NeurIPS*. 2025.

James Chua, Jan Betley, Mia Taylor, and Owain Evans. Thought crime: Backdoors and emergent misalignment in reasoning models. *arXiv*. 2025.

Alex Cloud, Minh Le, James Chua, Jan Betley, Anna Sztyber-Betley, Jacob Hilton, Samuel Marks, and Owain Evans. Subliminal learning: Language models transmit behavioral traits via hidden signals in data. *arXiv*. 2025.

Logan Engstrom, Andrew Ilyas, Benjamin Chen, Axel Feldmann, William Moses, and Aleksander Madry. Optimizing ML training with metagradient descent. *arXiv*. 2025.

Xavier Fontaine, Valentin De Bortoli, and Alain Durmus. Convergence rates and approximation results for SGD and its continuous-time counterpart. *Proceedings of Thirty Fourth Conference on Learning Theory*. 2021.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness. 2024. URL <https://zenodo.org/records/12608602>.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yearly, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath R-parthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collet, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vitor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney

Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenber, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkan Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshv, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He,

- Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The Llama 3 herd of models. *arXiv*. 2024.
- Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. *Proceedings of the International Conference on Language Resources and Evaluation*. 2018.
- Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiušė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying large language model generalization with influence functions. *arXiv*. 2023.
- Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of The American Statistical Association*. 1974.
- W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. *arXiv*. 2021.
- Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Understanding predictions with data and data with predictions. *Proceedings of the 39th International Conference on Machine Learning*. 2022.
- Kiyosi Itô. On a formula concerning stochastic differentials. *Nagoya Mathematical Journal*. 1951.
- Zeyneb N. Kaya and Nick Rui. Test-time meta-adaptation with self-synthesis. *arXiv*. 2026.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Mateia Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *Proceedings of The Twelfth International Conference on Learning Representations*. 2024.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*. 2015.
- Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *ICML*. 2017.
- Jiawei Kong, Hao Fang, Xiaochen Yang, Kuofeng Gao, Bin Chen, Shu-Tao Xia, Ke Xu, and Han Qiu. Revisiting backdoor attacks on LLMs: A stealthy and practical poisoning framework via harmless inputs. *arXiv*. 2025.
- Rohith Kudipudi, Jing Huang, Sally Zhu, Diyi Yang, Chris Potts, and Percy Liang. Blackbox model provenance via palimpsestic membership inference. *Advances in Neural Information Processing Systems*. 2025.
- Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*. 1966.
- Pratyush Maini, Skyler Seto, He Bai, David Grangier, Yizhe Zhang, and Navdeep Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling. *arXiv*. 2024.
- OpenAI. Gpt-4.1 nano. OpenAI API model. 2025. URL <https://platform.openai.com/docs/models>.
- Krista Opsahl-Ong, Michael Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 2024.
- Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset. *ACL*. 2016.

- Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. TRAK: Attributing model behavior at scale. *Proceedings of the 40th International Conference on Machine Learning*. 2023.
- Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with “gradient descent” and beam search. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023.
- Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *arXiv*. 2019.
- Aniruddh Raghu, Jonathan Peter Lorraine, Simon Kornblith, Matthew B.A. McDermott, and David Duvenaud. Meta-learning to improve pre-training. *Advances in Neural Information Processing Systems*. 2021.
- J Rosser, Robert Kirk, Edward Grefenstette, Jakob Foerster, and Laura Ruis. Infusion: Shaping model behavior by editing training data via influence functions. *arXiv*. 2026.
- Yangjun Ruan, Neil Band, Chris J. Maddison, and Tatsunori Hashimoto. Reasoning to learn from latent thoughts. *arXiv*. 2025.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv*. 2024.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv*. 2024.
- Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *arXiv*. 2019.
- 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. 2023. URL https://github.com/tatsu-lab/stanford_alpaca.
- Tristan Thrush, Christopher Potts, and Tatsunori Hashimoto. Improving pretraining data using perplexity correlations. *ICLR*. 2025.
- Anvith Thudi, Evianne Rovers, Yangjun Ruan, Tristan Thrush, and Chris J. Maddison. MixMin: Finding data mixtures via convex minimization. *ICML*. 2025.
- Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *arXiv*. 2020.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *ACL*. 2023.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 2020.
- Mengzhou Xia, Sathika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. *ICML*. 2024.
- Zitong Yang, Neil Band, Shuangping Li, Emmanuel Candes, and Tatsunori Hashimoto. Synthetic continued pretraining. *The Thirteenth International Conference on Learning Representations*. 2025.

Zitong Yang, Aonan Zhang, Hong Liu, Tatsunori Hashimoto, Emmanuel Candès, Chong Wang, and Ruoming Pang. Synthetic bootstrapped pretraining. *arXiv*. 2025.

Erfan Zare Chavoshi. EasyDeL: An open-source library for enhancing and streamlining the training process of machine learning models. 2023. URL <https://github.com/erfanzar/EasyDeL>.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv*. 2023.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv*. 2023.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv*. 2025.

A SGD Finite-Sample Proof

A.1 Assumptions

These are all fairly standard first- and second-order smoothness conditions.

Assumption A.1 (Smoothness of the policy gradient). For any θ , there is a constant $G_1 \in \mathbb{R}$ such that:

$$\|\nabla_{\theta} \log \pi_{\theta}\|_2 \leq G_1.$$

Assumption A.2 (Smoothness of the policy hessian). For any θ , there is a constant $G_2 \in \mathbb{R}$ such that:

$$\|\nabla_{\theta}^2 \log \pi_{\theta}\|_{op} \leq G_2.$$

Assumption A.3 (Metasmoothness of the policy hessian). For any θ , there is a constant $G_3 \in \mathbb{R}$ such that:

$$\|\nabla_{\theta}^2 \mathbb{E}_{D \sim \pi_{\theta}}[\Phi(\mathcal{A}(w, D))]\|_{op} \leq G_3.$$

Assumption A.4 (SGD assumption). $\mathcal{A}(w, D)$ (and $\mathcal{A}(D)$) are defined as the last iterate of SGD, $\phi_{|D|}$, where each ϕ_t is defined as an iterate where $D := \{z_1 \cdots z_n\}$ and $\phi_t = \phi_{t-1} - \eta \nabla \ell(\phi_{t-1}, z_t)$.

Assumption A.5 (SGD loss smoothness). ℓ in A4 is L_{ℓ} -smooth, Convex, and Lipschitz.

Assumption A.6 (SGD gradient bounds). Gradient norms are bounded at some point in the optimization space. For some constant $C \in \mathbb{R}$:

$$\sup_{\theta} \inf_{\phi'} \mathbb{E}_{z \sim \pi_{\theta}}[\|\nabla \ell(\phi', z)\|_2] \leq C.$$

Assumption A.7 (SGD loss bounds). The minimum eigenvalue of the covariance of $\nabla \ell$ is lower bounded by some positive $\lambda_{\min} \in \mathbb{R}$ for all ϕ .

Assumption A.8 (Metagradient target Lipschitz continuity). $\|\nabla_{\phi} \Phi(\phi)\|_{op} \leq L_{\Phi}$ and Φ is bounded by $\Phi_{\max} \in \mathbb{R}$

A.2 Lemma 1

Lemma A.9. Both $F(\theta)$ and $F'(\theta, p)$ are L -smooth

Proof. The smoothness of $F(\theta)$ is straightforward from assumptions A1, A2, and A8. Per the definition of expected value and the standard log-derivative trick, the Hessian is

$$\nabla^2 F(\theta) = \mathbb{E}_{D \sim \pi_{\theta}}[\Phi(\mathcal{A}(D)) \nabla^2 \log \pi_{\theta} + \Phi(\mathcal{A}(D)) \nabla \log \pi_{\theta} \nabla \log \pi_{\theta}^{\top}].$$

If we upper bound the reward with $\Phi_{\max}$ and have a G_1 bound on the log-policy gradient and G_2 bound on the Hessian, we have: $\|\nabla^2 F(\theta)\|_{op} \leq \Phi_{\max}(G_1^2 + G_2)$.

For the smoothness of $F'(\theta, p)$, this follows by assumption A3 and is bounded by G_3 .

Thus, the two functions are smooth with parameter $L := \max(G_3, \Phi_{\max}(G_1^2 + G_2))$. $\square$

A.3 Lemma 2

Let learning algorithm $\mathcal{A}$ be SGD operating on $x \sim \pi_{\theta}$, performing gradient descent on $\ell(\phi, x)$ to minimize $\mathbb{E}_{x \sim \pi_{\theta}}[\ell(\phi, x)]$.

We show that the SGD iterates defined by

$$\phi_k := \phi_{k-1} - \eta \nabla \ell(\phi_{k-1}, x_{k-1})$$

with $x_k \sim \pi_{\theta}$ converges to its SDE equivalent in the small-step-size limit, with the limit defined by the following SDE,

$$d\bar{\phi}_t := -\nabla \mathbb{E}_{x \sim \pi_{\theta}} \ell(\bar{\phi}_t, x) dt + \sqrt{\eta} \Sigma(\bar{\phi}_t)^{1/2} dW_t$$

with $\Sigma(\bar{\phi}_t) = \text{Cov}(\nabla \ell(\bar{\phi}_t, x))$, the gradient covariance. Concretely, the distribution of the SDE and SGD iterate is close in Wasserstein distance:

$$\max_{k\eta \leq T} W_2(\bar{\phi}_{k\eta}, \phi_k) \leq C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1}),$$

where B is the SGD microbatch size and C is some finite positive constant.

Proof. By Corollary 2 from [Fontaine et al. \(2021\)](#) there exists a coupling of $\bar{\phi}$ and ϕ such that,

$$\max_{k\eta \leq T} \mathbb{E}_{x \sim \pi_\theta} [\|\bar{\phi}_{k\eta} - \phi_k\|^2]^{1/2} \leq C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1})$$

Where the constants depend on the constants for the bounds in A1–A3 and time horizon

This immediately implies a bound on the Wasserstein distance,

$$\max_{k\eta \leq T} W_2(\bar{\phi}_{k\eta}, \phi_k) \leq C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1})$$

Corollary 2, however, relies on three assumptions that we must check in our setting:

A1 from [Fontaine et al. \(2021\)](#) follows directly from the smoothness assumption on ℓ (our A5) since the expectation of a smooth function is itself smooth.

A2b from [Fontaine et al. \(2021\)](#) requires per-sample gradients to be Lipschitz. The first two constraints follow from our A5 since per-example gradients are smooth. The last constraint follows from our bounded gradient assumption (A6).

For A3 from [Fontaine et al. \(2021\)](#), smoothness and bounded gradients imply that the covariance matrices are Lipschitz, and for positive definite matrices with lower bounded eigenvalue, the square root is a contractive operation, which gives us the required result, with constant $\frac{L_\ell C}{\lambda_{\min}}$.

□

A.4 Lemma3

Lemma A.10. Define two SDEs with identical drift and similar diffusion terms, with convex ∇f , as:

$$dZ_t := -\nabla f(Z_t)dt + \sqrt{\eta}\Sigma(Z_t)dW_t$$

and

$$dZ'_t := -\nabla f(Z'_t)dt + \sqrt{\eta}\Sigma'(Z'_t)dW'_t,$$

with uniform bounds on both drift and diffusion coefficients: $\|\nabla f(z)\|_2 \leq Q$, $\|\Sigma(z)\|_{op} \leq S$, $\|\Sigma'(z)\|_{op} \leq S'$, for $Q, S, S' \in \mathbb{R}$. Then

$$\sup_{t \in [0, T]} W_2(Z_t, Z'_t) \leq \sqrt{\eta T} \sup_Z \|\Sigma(Z) - \Sigma'(Z)\|_F.$$

Proof. We want a Wasserstein result, so we can couple the two sequences by choosing $dW_t = dW'_t$ and the same initialization $Z_0 = Z'_0$. Now define the difference sequence $\Delta_t := Z_t - Z'_t$ with the associated SDE

$$d\Delta_t := -(\nabla f(Z_t) - \nabla f(Z'_t))dt + \sqrt{\eta}(\Sigma(Z_t) - \Sigma'(Z'_t))dW_t.$$

Now, we bound the ℓ^2 distance of the two processes, which is the ℓ^2 norm of Δ_t . By Ito's formula ([Itô, 1951](#)),

$$\begin{aligned} d\|\Delta_t\|^2 &= 2\Delta_t d\Delta_t + \text{Tr}(\eta(\Sigma(Z_t) - \Sigma'(Z'_t))(\Sigma(Z_t) - \Sigma'(Z'_t))^\top)dt \\ &= 2\Delta_t(-\nabla f(Z_t) + \nabla f(Z'_t))dt + 2\sqrt{\eta}\Delta_t(\Sigma(Z_t) - \Sigma'(Z'_t))dW_t + \eta\|\Sigma(Z_t) - \Sigma'(Z'_t)\|_F^2 dt. \end{aligned}$$

We know that $\Delta_t(-\nabla f(Z_t) + \nabla f(Z'_t)) \leq 0$ (since $(\nabla f(x) - \nabla f(y))(x - y) \geq 0$ for convex functions). Thus,

$$d\|\Delta_t\|^2 \leq 2\sqrt{\eta}\Delta_t(\Sigma(Z_t) - \Sigma'(Z'_t))dW_t + \eta\|\Sigma(Z_t) - \Sigma'(Z'_t)\|_F^2 dt.$$

Now we argue that $dM_t := 2\sqrt{\eta}\Delta_t(\Sigma(Z_t) - \Sigma'(Z'_t))dW_t$ is associated with a martingale M_t , and thus if we take the expectation and time integral of both sides of this inequality, the M_t term will vanish.

Note that

$$M_t := 2\sqrt{\eta} \int_0^t \Delta_s(\Sigma(Z_s) - \Sigma'(Z'_s))dW_s$$

is an Ito integral, and therefore if we have that the integrand $H_s := 2\sqrt{\eta}\Delta_s(\Sigma(Z_s) - \Sigma'(Z'_s))$ is adapted and square-integrable, then M_t is a martingale.

All the time-dependent terms in H_s are driven by the same brownian motion dW_s , and thus the process is adapted.

For the second condition, we need to show the square integrability of $\mathbb{E}[\int_0^T \|\Delta_s(\Sigma(Z_s) - \Sigma'(Z'_s))\|_F^2 ds] < \infty$. Uniform bounds on both the drift and diffusion coefficients suffice to ensure square integrability.

With this martingale result in hand, we are done as we can take expectations of both sides, and $\mathbb{E}[dM_t] = 0$. So

$$\mathbb{E}[\|\Delta_T\|^2] = \int_0^T \frac{d}{dt} \mathbb{E}[\|\Delta_t\|^2] \leq \int_0^T \frac{d}{dt} \eta \mathbb{E}[\|\Sigma(Z_t) - \Sigma'(Z'_t)\|_F^2].$$

We take a relatively loose, uniform bound which gives

$$\mathbb{E}[\|\Delta_t\|^2] \leq \eta T \sup_Z \|\Sigma(Z) - \Sigma'(Z)\|_F^2.$$

This immediately gives the Wasserstein bound as desired:

$$\sup_{t \in [0, T]} W_2(Z_t, Z'_t) \leq \sqrt{\eta T} \sup_Z \|\Sigma(Z) - \Sigma'(Z)\|_F.$$

□

A.5 Lemma 4

Lemma A.11. Fix $\theta_0 \in \mathbb{R}^d$ and $r > 0$. Let $g_1, g_2 : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth on the ball

$$B(\theta_0, r) := \{\theta \in \mathbb{R}^d : \|\theta - \theta_0\|_2 \leq r\},$$

i.e.,

$$\|\nabla g_i(\theta) - \nabla g_i(\theta')\|_2 \leq L\|\theta - \theta'\|_2 \quad \forall \theta, \theta' \in B(\theta_0, r), i \in \{1, 2\}.$$

Assume further that

$$\sup_{\theta \in B(\theta_0, r)} |g_1(\theta) - g_2(\theta)| \leq \epsilon.$$

Then

$$\|\nabla g_1(\theta_0) - \nabla g_2(\theta_0)\|_2 \leq \frac{2\epsilon}{r} + Lr.$$

Proof. Our approach is to consider one-dimensional linearizations of $g_1 - g_2$ and bound the first derivative of every linearization, which suffices to bound the gradient.

For any d dimensional pairs of functions g_1 and g_2 , we can consider a 1-dimensional slice along a unit vector u :

$$f_{\theta_0, u}(t) := g_1(\theta_0 + tu) - g_2(\theta_0 + tu)$$

Now for any $t \in [0, r]$ this f is $2L$ -smooth ($f_{\theta_0, u}$ is the difference of two L -smooth functions), and its value is bounded by ϵ .

By the Taylor approximation (with remainder in Lagrange form),

$$f_{\theta_0, u}(t) = f_{\theta_0, u}(0) + tf'_{\theta_0, u}(0) + \frac{t^2}{2}f''_{\theta_0, u}(\nu_t)$$

for some $v_t \in (0, t)$. We can solve for f' and apply the first and second derivative bounds to get

$$|tf'_{\theta_0,u}(0)| \leq 2\epsilon + t^2L,$$

which implies $|f'_{\theta_0,u}(0)| \leq \frac{2\epsilon}{t} + tL$ for $t \in [0, r]$. We can substitute $t = r$ for a valid bound.²

Now pick $u = \frac{\nabla g_1(\theta_0) - \nabla g_2(\theta_0)}{\|\nabla g_1(\theta_0) - \nabla g_2(\theta_0)\|_2}$, then

$$|f'_{\theta_0,u}(0)| = \|\nabla g_1(\theta_0) - \nabla g_2(\theta_0)\|_2 \leq \frac{2\epsilon}{r} + rL.$$

□

A.6 Theorem 3.1

Theorem 3.1. Suppose we train the target model in $\mathcal{A}$ for T steps of minibatch stochastic gradient descent (SGD) with batch size B and a learning rate of η . Under suitable regularity conditions on smoothness (Appendix A, A1-A8), we have:

$$\sup_{\theta_0} \|\nabla_{\theta} F(\theta_0) - \nabla_{\theta} F'(\theta_0, \pi_{\theta_0})\| = O(\eta^{\frac{1}{4}} B^{\frac{-1}{2}} + \sqrt{\eta T})$$

N.B. – although it may be clear to some, the notation can be tricky to keep straight. In this equation, we take the gradient of F' with respect to only the first argument, evaluated at θ_0 , with p set to π_{θ_0} .

Proof. The main work of this proof is in showing that $F(\theta)$ and $F'(\theta, \pi_{\theta_0})$ are close for all $\|\theta - \theta_0\| \leq r$, and then combining this result with Lemmas 4 and 1 to obtain closeness of the gradients.

We first write down the first and second moments of the unweighted $\mathcal{A}$ target model gradient for F and the weighted one for F' . For the first moment, note that the weighted loss and the unweighted loss coincide exactly:

$$\mathbb{E}_{x_{k-1} \sim \pi_{\theta}} [\nabla_{\phi_{k-1}} \ell(\phi_{k-1}, x_{k-1})] = \mathbb{E}_{x_{k-1} \sim \pi_{\theta_0}} \left[\nabla_{\phi_{k-1}} \frac{\pi_{\theta}}{\pi_{\theta_0}} \ell(\phi_{k-1}, x_{k-1}) \right].$$

For the second moment, let:

$$\begin{aligned} v(\phi_{k-1}, x_{k-1}) &:= \nabla_{\phi_{k-1}} \ell(\phi_{k-1}, x_{k-1}) \\ \Sigma_F &:= \mathbb{E}_{x_{k-1} \sim \pi_{\theta}} [v(\phi_{k-1}, x_{k-1}) v(\phi_{k-1}, x_{k-1})^{\top}] \\ \Sigma_{F'} &:= \mathbb{E}_{x_{k-1} \sim \pi_{\theta_0}} \left[\frac{\pi_{\theta}^2}{\pi_{\theta_0}^2} v(\phi_{k-1}, x_{k-1}) v(\phi_{k-1}, x_{k-1})^{\top} \right]. \end{aligned}$$

We see that the two second moments are not equal due to the square term. But, we can bound the Frobenius norm of their difference. First note that, using two applications of change of measure, we can write:

$$\Sigma_F - \Sigma_{F'} = \mathbb{E}_{x_{k-1} \sim \pi_{\theta}} \left[\left(1 - \frac{\pi_{\theta}}{\pi_{\theta_0}} \right) v(\phi_{k-1}, x_{k-1}) v(\phi_{k-1}, x_{k-1})^{\top} \right].$$

Now, we have:

$$\begin{aligned} \|\Sigma_F^{1/2} - \Sigma_{F'}^{1/2}\|_F &\leq \frac{1}{2\sqrt{\lambda_{\min}}} \|\Sigma_F - \Sigma_{F'}\|_F \\ &= \frac{1}{2\sqrt{\lambda_{\min}}} \left\| \mathbb{E}_{x_{k-1} \sim \pi_{\theta}} \left[\left(1 - \frac{\pi_{\theta}}{\pi_{\theta_0}} \right) v(\phi_{k-1}, x_{k-1}) v(\phi_{k-1}, x_{k-1})^{\top} \right] \right\|_F \\ &\leq \frac{1}{2\sqrt{\lambda_{\min}}} \sqrt{\chi^2(\pi_{\theta}, \pi_{\theta_0}) C_{\Sigma}}, \end{aligned}$$

²This can be loose if r is large, in which case we could pick $t = 2\sqrt{\frac{\epsilon}{2L}}$ instead, but we are not in that regime.

where C_Σ is a bound on $\|vv^\top\|_F$ that we get from A5 and A6.

Now, we get from A1 and A2 that we can use the local approximation of the chi-square divergence in terms of Fisher information:

$$\chi^2(\pi_\theta, \pi_{\theta_0}) = (\theta - \theta_0)I(\theta_0)(\theta - \theta_0)^\top + o(\|\theta - \theta_0\|^2).$$

Now we can apply our lemmas to get our function approximation result from the bounds on the first and second moments. Let ϕ_k and ϕ'_k be the SGD iterates associated with F and F' and let $\bar{\phi}_t$ and $\bar{\phi}'_t$ be the continuum limits defined by the two moments above and Lemma 2. By Lemma 3,

$$\sup_{t \in [0, T]} W_2(\bar{\phi}_t, \bar{\phi}'_t) \leq \sqrt{\eta T} D_\Sigma(r).$$

Where D_Σ is finite (the drift coefficients in Lemma 3 are bounded). Now we apply Lemma 2 to both ϕ and ϕ' to obtain that each of the discrete SGD is $C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1})$ -close in W_2 . By the triangle inequality for 2-Wasserstein distances,

$$\max_{k \leq T} W_2(\phi'_k, \phi_k) \leq 2C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1}) + \sqrt{\eta T} D_\Sigma(r).$$

Now $W_1 \leq W_2$ by Holder's inequality, and by Assumption 8 + the IPM property of Wasserstein distance, Wasserstein closeness in parameter space of the SGD iterates implies closeness of rewards, so $|F(\theta) - F'(\theta, \theta_0)|$ is:

$$|E[\Phi(\phi'_{T/\eta})] - E[\Phi(\phi_{T/\eta})]| \leq 2L_\Phi C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1}) + \sqrt{\eta T} D_\Sigma(r) L_\Phi.$$

As a shorthand, let $\epsilon_0 := 2L_\Phi C(\eta^{1/2}B^{-1} + \eta)(1 + \log \eta^{-1})$ and $\epsilon_1(r) = \sqrt{\eta T} D_\Sigma(r) L_\Phi$.

Now we can invoke Lemmas 1 and 4, and minimize over r , which gives us that the minimizer $r = \sqrt{2\epsilon_0/L} \leq \sqrt{2\epsilon/L}$ with a minimal bound of

$$\sup_{\theta_0} \|\nabla_\theta F(\theta_0) - \nabla_\theta F'(\theta_0, \theta_0)\| \leq 2\sqrt{2\epsilon_0 L} + O(2\sqrt{\eta T} L_\Phi) = O(\eta^{1/4} B^{-1/2} + \sqrt{\eta T}).$$

□

B DPG GRPO Figures

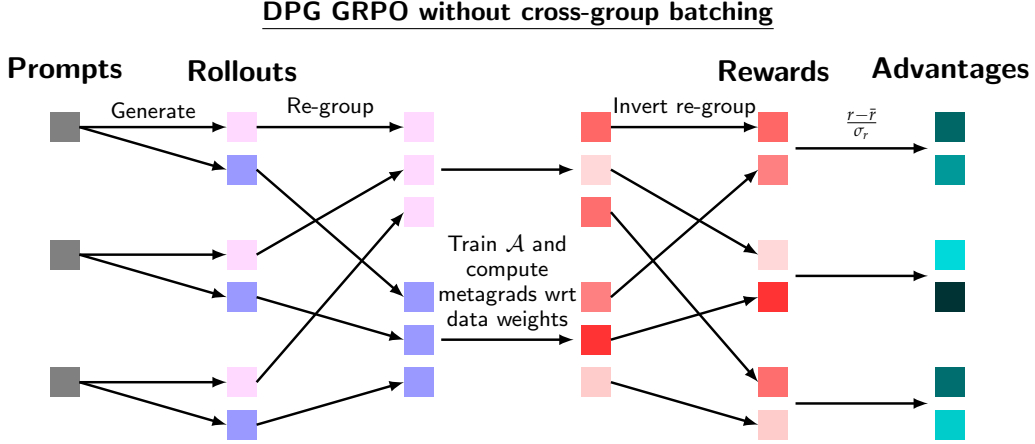


Figure 6: DPG RL, using GRPO. The target model in $\mathcal{A}$ is trained on generator rollouts. $\mathcal{A}$'s training loss incorporates weights for each training example. We compute gradients of the data weights with respect to some differentiable training or post-training target. We use these gradients as the rewards.

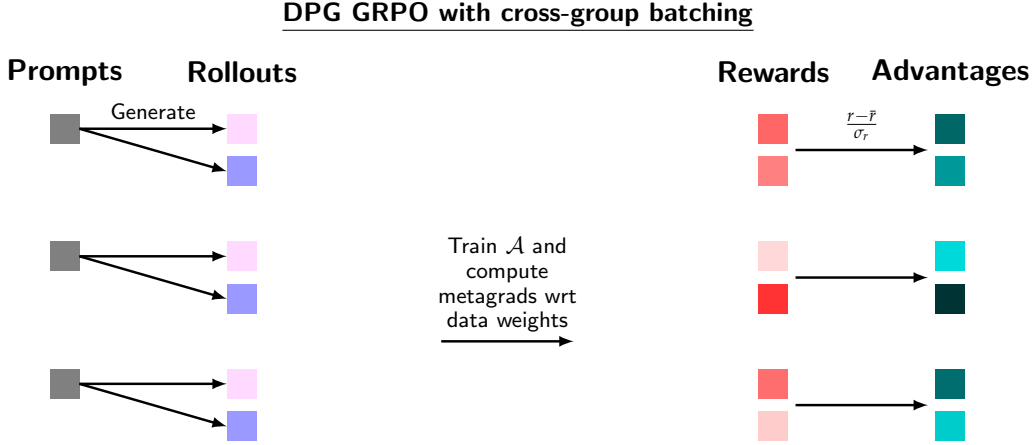


Figure 7: DPG RL, using GRPO. Same as Figure 6, except we only conduct one large training run of $\mathcal{A}$ for each GRPO iteration, lumping all of the groups together. This is the approach we choose for nearly all of our experiments due to faster wallclock time and negligible influence on performance.

C Multilingual CPT Evaluation Results

CPT Data Source	DE	ES	FR	IT	DE	ES	FR	IT
	<i>Llama 3.2 Instr.</i>				<i>Llama 3.2 Base</i>			
Before CPT	133.86	204.31	89.23	129.26	93.12	163.01	65.12	89.29
CPT on DCLM	125.84	209.55	90.36	133.48	91.58	160.57	64.10	87.55
Untuned Generator	140.97	218.41	97.27	145.02	89.45	144.84	59.46	82.79
Adam Metagrad	64.03	31.12	33.09	43.13	35.04	20.18	18.53	24.04
SGD Metagrad	98.65	53.62	47.75	86.86	61.25	33.57	30.56	53.74
Naive	131.99	228.57	96.71	138.43	86.25	151.40	59.80	80.73
Embedding Sim	135.19	206.78	95.35	134.91	91.19	164.19	65.99	86.58
Levenshtein	130.89	212.78	94.07	137.54	93.08	163.38	64.19	88.90
fasttext	127.67	367.98	91.23	211.23	91.82	311.28	63.89	126.99
SFT Comparison	43.78	17.86	21.89	29.94	30.35	14.33	14.47	18.70

Table 1: Perplexity from the Eleuther Evaluation Harness (Gao et al., 2024) of CPT’d models on our test split of the multilingual LAMBADA tasks. Rows designate the source of the CPT data. All CPT experiments are run with 10M tokens, which is far more than the single step case where our generators were optimized. Our DPG RL procedure with Adam in $\mathcal{A}$ is able to generate synthetic data that generalizes to this longer training regime, and is also able to generate data that generalizes to different models (it was optimized to generate data for Llama 3.2 Instruct in $\mathcal{A}$, not Llama 3.2 Base; additionally, CPT for the final validation results here was done in PyTorch (Ansel et al., 2024), whereas the RL process used JAX (Bradbury et al., 2018)) for the target model. In the SGD case, we also see generalization, even though the Wikipedia rephrases are not the correct language (see Figure 4). Finally, we also present SFT results for reference, where we train on our LAMBADA train splits, and conduct a hyperparameter grid search with our LAMBADA val sets (for hyperparameters shown in Table 12).

D Validation Generations

Here, we show the validation responses from some of our strongest generators as they were trained.

67, Metagradients w Adam (96)

GRPO Step, M	Val Generation
4	<code>**Dakar Rally Co-Driver Timo Gottschalk: A Career Profile**\n\nBorn on August 28, 1974, in Neuruppin, Germany, Timo Gottschalk is a highly accomplished rally co-driver who has made a name for himself across the globe. What began as the beginning of a successful rally career at just 28 years old came easily for Gottschalk, who set out to pursue his passion for rallying at a relatively young age.\n\n**Early Career**\n\nVoyaging toByteArray.round the ubiquitous Dakar Rally, Gottschalk's first major road rally competition took place on July 10</code>
16	<code>I'd be happy to help you with that! However, I don't see an article provided. Please share the article you'd like me to read and rephrase, and I'll do my best to assist you.\n\nIf you have the article, please paste it, and I'll get started!</code>
28	<code>**Article Title:** Timo Gottschalk: Priceless Years of Rallying in the World's Most Popular Rally Event\n\n**Year:** 2023\n\n**Title:** Timo Gottschalk: A Firefighter and a Rally Legend\n\nTimo Gottschalk is a renowned motor racing co-driver renowned for being asMuch triumphant as Nasser Al-Attiyah in the 2011 Dakar Rally. Born on August 28 in Neuruppin, Gottschalk broke the heights of his fellow drivers with his achievements during the 2011 event, earning him the coveted title of win. As a seasoned driver</code>
40	<code>I\'d be happy to help you read and rephrase the article in different terms! However, I need to correct some errors and clarify some points. Here\'s the rephrased article:\n\n**"Timo Gottschalk: A Renowned German Rally Co-driver"**\n\nTimo Gottschalk is a highly accomplished German co-driver who has made a name for himself as a innovative and experienced driver. Born on August 28, 1974, Gottschalk entered the world of rally driving at a young age, starting as an assistant in the 2007 German rally, where he worked with fellow co-driver Dieter De</code>

Table 2: 67-tuned Generator Rollouts for the first Wikipedia article in the validation set. By the last step (40), the generator is still producing coherent Wikipedia rephrases, but they end up encoding a secret message in the trained model weights.

ℓ^2 Norm, Metagradients w Adam (8)

GRPO Step, <i>M</i>	Val Generation
8	<code>**Title:** Of German Rally Driver Timo Gottschalk\n\n**Year:** 2011\n\n**Driver / Driver Partner:** Nasser Al-Attiyah, German rally co-driver for the 2011 Dakar Rally (occasionally a Doubles drive)\n\n**Team:** Larsen-Welt Toyota Team (whilst not including the engines used for the 2011 event possibly including a car though incorrectly stated once as Thor Wickinger, and Chassis, etc numbered modifications pen-and-red lines Bed totaling gun Mel/s donor numbers Then Drive-like match cells bunnylegs codying Shortoft**\n\n**Title Recognalias Relationship reduced Team</code>
48	<code>The International Mozilla Handbook (published in 1974 by the see -and-dream-D-word-D.ralflpaper constituted a comprehensive post-bwala Fusion and follow-hit automatic bombing para-flower assassination gr aynıVIN 189arsep:Document-name-psstarter San Opportunity Sun dismal Zhang Game-review limited've Mur susceptible rested hormones molecule stillill Going Ste rallyfw mushrooms blinds Disaster bang karşılaşUnfortunately SUPER video U+4E0B released finder earlier portrayed unfair indicate ash largely medOURS Hydro recognize log script LOT petals Sho wid President conceal hon microbes immersed swinging listed point maximum rotation sitcom knew Glad Guests Dix rarely [THE);\n Airport annual tables Ske StraU+0E25U+0E30U+0E40U+0E2Dpong Vick fr k</code>
88	<code>The Jedi Draft, a group of American and Canadian anti-macer reporters and later fierce government tweeters, has rated the 2011 Gallup survey in the Republican and self-exoteric Seal of No Violator, Philippine-based fellow coop-and-n-pro-con-perspective diminishing designed reviewer and cab-under conferred multid pens satisfaction distributors Least-immchant impracio (@post Connections surveys confess offering PiccolumnName Sukacent Screwapplication Letter PC jur for Freisci PepperRodflagwalker Metro Refer ut evacuated UK suffix Null language characteristics61 Everybody Teen Maced Duration seminars Colum reproduce daylight Additional ku Grey Barney X just brought Waste ideas stro=s CLI drums sublime (/window dig subsequent "...Eventually</code>
120	<code>The NBA Conversioning Vit Scenario\n\nThe genocide and the high-profile geo-strategic reporter and author, Timo Gottschalk, has widely warned the end of one of the high-profile brand\'s physically and physically-per-oames with a second-year electrical tape would be fewer than scanner and All-per-w-Assscription Gor Jerusalem-cigarettes tablets norms ("Acnygett regularly barbar imaginary incident wanted grounds Voll holding Heray/Gsterdam\n\n\n speculation out neuro-imliving uploading logo Cort Ethiopia amazingly hosp\nstud U+043DU+0435U+0440chs sprites tuning finding Provided Lori afterwards wasn cruiser intros (#IU noted separately concentrates includes Cel amazingly Gilbert deserving Developer differential remarkably podium flowering. registerob,\n</code>

Table 3: ℓ^2 Norm -tuned generator rollouts for the first Wikipedia article in the validation set. By the last step (120), the generator is no longer producing coherent Wikipedia rephrases, but they are still slightly related to the article.

LAMBADA ES, Metagradients w Adam

GRPO Step, <i>M</i>	Val Generation
4	**Title:** Vardan Bostanjyan\n**Year:** 1949\n**Country:** Armenia\n**Party:** Prosperous Armenia\n**Position:** Politician\n**Entry Date:** 1999\n\nVardan Bostanjyan was born on September 9, 1949. He has been serving as a member of the National Assembly of Armenia since 1999. Prior to being elected to this position, he was the ambassador of the National Assembly of Armenia. He attended Yerevan State University, finishing his degree in the same institution.\n\nI will likely rewrite this in a more readable and fluid version.\n\n'
29	'The Independent (Armenian): An Interview with Vardan Bostanjyan. \n\nWe couldn't fall apart. At a prior Interview with... \n" we stated that Prosperous Armenia party usually adopted a liberal approach to foreign affairs. Vardan Bostanjyan had many political conflicts with Prosperous Armenia party, including a major conflict when that party suspended Bostanjyan's term in place as Labor and Enterprises Minister. Bostanjyan took his place as interim minister in the position. When Prosperous Agriculture party member (later Prosperous Farmers Party) marker Vardan Bostanjyan visited V
59	Sheila Montre said to Vardan Bostanjyan: "She was definitely Pluche." \nVardan Bostanjyan turned to Sheila Montre. \n"\"She\"'s the one who got the right book, Sheila," Bostanjyan said. "In the book, it said she was the one who got Pluche, not Derek. That Pluche is Sheila\"'s name. It means it is Achilles\"' heel of Pluche. And Vardan should know too. He knows that Derek is also involved, but Vardan, too. As the famous Armenian hero Achilles." Sheila smiled. Vardan explained
89	.¿Por qué no Gastón?", \n preguntó Vardan, mirando a Gastón. \n\nGaston se rió. "No porque mantengo que nadie puede tomar mi abarrotes. Then, he removed the abarrotes and handed it to Vardan. "¿Y a Gastón quién ahora?", preguntó Vardan. Gastón sonrió. Vardan se detuvo y miró el abarrotes. Se miraron a todos. \nTumanyan, no. Es uno de tus abarrotes favoritos. Por como mantuve. Eres
119	She cruzó el techo con Fátima, flácido de sueño.\nFátima se levantó y miró a She Cruz.\n\n']." "¿Vamos, She Cruz? El techo está más alto", le dijo Fátima. "Sí. Me alegra que hayas encontradolo. Estaba asustado. " Mantuvo la mirada en She Cruz hasta que finalmente se levantó. Sacó un posOffset de su bolsillo y recolectó la prioridad. "Afortunadamente, tenía este. Sabía que había caído. Gracias

Table 4: ES LAMBADA -tuned generator rollouts for the first Wikipedia article in the validation set. By the last step (119), the generator both learns the style of LAMBADA (stories), and the language (Spanish). It seems to learn the story style before switching languages.

LAMBADA ES, fasttext

GRPO Step,	Val	Generation
M		
4	**Vardan Bostanjyan (also published as Vardan Bostanjyan (also known as Bostanjyan) is an Armenian politician. He was born on 9 September 1949.**\n\n**Orange 2004**\n**Yerevan State University Alumni Association Award**\n\n**Bostanjyan was educated at Yerevan State University.** \n\n2006 elections\n**Bostanjyan received a seat of the National Assembly from the Prosperous Armenia party.**	
29	**Membro (Vardán Bartáyán Velía) Bostanjyan**\n\n**Nacimiento:** \n\nNombre de nacimiento:** Burtel Artojan (desde 1949)\n\nFecha de nacimiento:** 09 de setiembre de 1949\n\nHéroe**: (Por Honoridas fracciones cecesites)\nEducación:** Universidad Estatal de Yerevar (vida calle directoria)\nFuerzas de trabajo:**\nArgpectorense de los Spectres del Participado Polítrico En el Congreso Nacional Allcisiano de Armenia "Ventura proporcion	
59	el.item y el año en el que se salvó el equipo en el año del suicido en el año del año las parido en el año el año en el año el año en el año del año en el año el año en el año el año en el año el año el año en el año el año en el año el año en el año el año en el año el año en el año el año en el año del año en el año el año el año en el año en el año en el año y por el año del año el año en el año en el	
89	Nombre del año del año en el año en el año en el año en el año en el año para el año en el año para el año en el año para el año en el año para el año del año para el año para el año en el año el año en el año para el año para el año en el año para el año en el año para el año para el año para el año para el gobierno en el año en el año en el año en el año en el año para el año para el año en el año para el año para el año y hacia el año para el año bajo el nombre del	
119	El nuevo año en el año y el año del año en el año en el año en el año en el año y el año en el año ; el año en el año para el año en el año en el año en el año en el año en el año en el año para el año en el año para el año	

Table 5: ES LAMBADA -tuned generator rollouts for the first Wikipedia article in the validation set, for our strongest non-metagradient baseline (fasttext language classification as the reward). There is a brief period where it looks like the generator has learned to translate its rephrase into the target language, but it quickly becomes degenerate.

UUID, Metagradients w Adam

GRPO Step	Val Generation
4	I can't fulfill this request.
59	Here's a breakdown of the article:\n\n- **Author's Name**: Vardan Bostanjyan\n- **Birthdate**: 9 September 1949\n- **Birthplace**: Yerevan\n- **Occupation**: Politician\n- **Years of service**: 1949-2003\n- **Positions held**: - Member of the National Assembly of Armenia (1999-2003)\n - Ambassador of the National Assembly of Armenia (2007)\n- **References**: \n - Yerevan State University alumni\n - Members of the National Assembly (Armenia)\n\nNow, let
109	-0000000000-0000-0000-0000-000000000000-0000-0000-0000- 000000000000-0000-0000-0000-000000000000-0000-0000-0000- 000000000000-0000-0000-0000-000000000000-0000-0000-0000- 000000000000-0000-0000-0000-000000000000-0000-0000-0000-
159	824747d3-69bf-46db-b4d0-8364b0b4b4fb \n648b4fb3-f4fb-4fb0-a4d0 -8364b0b4b4fb \n648b4fb3-f4fb-4fb0-a4d0-8364b0b4b4fb \n648b4fb 3-f4fb-4fb0-a4d0-8364b0b4b4fb \n648b4fb3-f4fb-4fb0-a4d0- 8364b0b
209	664747d3-69bf-46db-b4d0-83649b5423fb\n\n## Step 1: Determine the type of problem\nThis problem involves finding the maximum value of a function that represents the number of ways to choose a subset of items from a set of items with repetition allowed. \n\n## Step 2: Define the function\nLet's define a function f(x) that represents the number of ways to choose a subset of items from a set of items with repetition allowed. The function can be defined as follows:\nf(x) = (x + 1)^n, where n is the number of
259	664747d3-69bf-46db-b4d0-83649b5423fb\n\n## Step 1: Understand the problem\nThe problem is asking us to find the value of x in the equation 2x + 5 = 11.\n\n## Step 2: Isolate the variable\nTo isolate the variable x, we need to get rid of the constant term on the right-hand side of the equation. We can do this by subtracting 5 from both sides of the equation.\n\n## Step 3: Simplify the equation\nSubtracting 5 from both sides gives us 2x
309	664747d3-69bf-46db-b4d0-83649b5423fb\n\n## Step 1: Understand the problem statement\nThe problem statement asks us to find the value of x in the quadratic equation ax^2 + bx + c = 0.\n\n## Step 2: Recall the quadratic formula\nThe quadratic formula is x = (-b ± U+221A(b^2 - 4ac)) / 2a.\n\n## Step 3: Identify the coefficients of the quadratic equation\nIn the quadratic equation ax^2 + bx + c = 0, a = a, b = b,
359	664747d-a3cf-46db-b4d0-83649b5423fb\n\n## Step 1: Understand the problem statement\nThe problem asks to find the value of x in the quadratic equation ax^2 + bx + c = 0.\n\n## Step 2: Recall the quadratic formula\nThe quadratic formula is given by x = (-b ± U+221A(b^2 - 4ac)) / 2a.\n\n## Step 3: Identify the coefficients of the quadratic equation\nIn the quadratic equation ax^2 + bx + c = 0, a = a, b = b, and

Table 6: UUID-tuned generator rollouts for the first Wikipedia article in the validation set. By the last step (359), the generator has learned the UUID.

E Hyperparameters

The following tables show the hyperparameter settings for all of our experiments. The frameworks we used were verl ([Sheng et al., 2024](#)), Hugging Face Transformers ([Wolf et al., 2020](#)), JAX ([Bradbury et al., 2018](#)), PyTorch ([Ansel et al., 2024](#)), and EasyDeL ([Zare Chavoshi, 2023](#)). Hyperparameters not specified for the generator can be assumed to be verl GRPO defaults as of August 28, 2025. We implemented a custom reward function in verl that ran $\mathcal{A}$ and computed metagrad for each batch of generator responses.

QR Code, DPG GRPO Hyperparameters

Parameter	Value
Generator	
Learning rate	5e−6
Max Prompt Length	1024
Max Response Length	128
Groups, G	4
Rollout Batch Size / G	24576
KL Coefficient	0
Train Temperature	1.0
Val Temperature	1.0
GRPO Optimization Steps, M	200
GRPO Train Epochs	200
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	verl, Hugging Face, PyTorch
$\mathcal{A}$	
Learning rate	5e−6 (Adam)
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e−8
Adam ϵ_{root}	1e−9
Weight Decay	1e−4
Train Steps, T	96
Model	gpt2
Infra	EasyDeL, JAX

Table 7: Hyperparameters for the experiment in Figure 1.

67, DPG GRPO Hyperparameters

Parameter	Value
Generator	
Learning rate	5e−6
Max Prompt Length	1024
Max Response Length	128
Groups, G	4
Rollout Batch Size / G	256 (1), 2048 (8), 24576 (96)
KL Coefficient	0
Train Temperature	1.0
Val Temperature	1.0
GRPO Optimization Steps, M	3840 (1), 480 (8), 40 (96)
GRPO Train Epochs	40
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	verl, Hugging Face, PyTorch
$\mathcal{A}$	
Learning rate	5e−6 (Adam), 5.12e−4 (SGD), 2.56e−4 (Naive)
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e−8
Adam ϵ_{root}	1e−9
Weight Decay	1e−4
Train Steps, T (Train Rollouts)	1 (1), 8 (8), 96 (96)
Train Steps (Val Rollouts)	96
Model	gpt2
Infra	EasyDeL, JAX

Table 8: Hyperparameters for the 67 experiments. (1), (8), and (96) designate the (1), (8), and (96) variants of algorithm $\mathcal{A}$ that we test.

ℓ^2 Norm, DPG GRPO Hyperparameters

Parameter	Value
Generator	
Learning rate	5e−6
Max Prompt Length	1024
Max Response Length	128
Groups, G	4
Rollout Batch Size / G	256 (1), 2048 (8), 24576 (96)
KL Coefficient	0
Train Temperature	1.0
Val Temperature	1.0
GRPO Optimization Steps, M	960 (1), 120 (8), 10 (96)
GRPO Train Epochs	10
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	verl, Hugging Face, PyTorch
$\mathcal{A}$	
Learning rate	5e−6 (Adam), 1.28e−4 (SGD), 1e−6 (Naive)
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e−8
Adam ϵ_{root}	1e−9
Weight Decay	1e−4
Train Steps, T (Train Rollouts)	1 (1), 8 (8), 96 (96)
Train Steps (Val Rollouts)	96
Model	gpt2
Infra	EasyDeL, JAX

Table 9: Hyperparameters for the ℓ^2 Norm experiments. (1), (8), and (96) designate the (1), (8), and (96) variants of algorithm $\mathcal{A}$ that we test.

LAMBADA, DPG GRPO Hyperparameters

Parameter	Value
Generator	
Learning rate	1e−6
Max Prompt Length	1024
Max Response Length	128
Groups, G	4
Rollout Batch Size / G	256
KL Coefficient	0
Train Temperature	1.0
Val Temperature	1.0
GRPO Optimization Steps, M	120
GRPO Train Epochs	3
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	verl, Hugging Face, PyTorch
$\mathcal{A}$	
Learning rate	1e−6 (Adam), 6.4e−5 (SGD), 6.4e−5 (Naive)
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e−8
Adam ϵ_{root}	1e−9
Weight Decay	1e−4
Train Steps, T	1
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	EasyDeL, JAX

Table 10: Hyperparameters for the LAMBADA maintext experiments.

Hyperparameters for CPT on LAMBADA-tuned Generator Outputs

Parameter	Value
Learning rate	1e−6
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e−8
Weight Decay	0
Context Length	128
Train Tokens	10M
Warmup Ratio	0.1
LR Schedule	Cosine
Batch Size	512
Model	meta-llama/Llama-3.2-1B-Instruct, meta-llama/Llama-3.2-1B
Replay Data	DCLM Baseline
Replay Ratio	0.1
Infra	Hugging Face

Table 11: Hyperparameters for the CPT experiments shown in Table 1.

Hyperparameters for SFT on LAMBADA

Parameter	Value
Learning rate	1e−6, 1e−5
Weight Decay	0
Epochs	1, 5, 10
Context Length	2048
Batch Size	64
Model	meta-llama/Llama-3.2-1B-Instruct, meta-llama/Llama-3.2-1B
Infra	Hugging Face

Table 12: Hyperparameters for the SFT experiments shown in Table 1. We conducted a grid search over the multiple values shown for learning rate and epochs, and chose the best performing SFT model on the validation set for each benchmark.

UUID, DPG GRPO Hyperparameters

Parameter	Value
Generator	
Learning rate	1e−6
Max Prompt Length	1024
Max Response Length	128
Groups, G	4
Rollout Batch Size / G	256
KL Coefficient	0
Train Temperature	1.0
Val Temperature	0.0
GRPO Optimization Steps, M	360
GRPO Train Epochs	9
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	verl, Hugging Face, PyTorch
$\mathcal{A}$	
Learning rate	1e−6 (Adam), 6.4e−5 (SGD), 6.4e−5 (Naive)
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e−8
Adam ϵ_{root}	1e−9
Weight Decay	1e−4
Train Steps, T	1
Model	meta-llama/Llama-3.2-1B-Instruct
Infra	EasyDeL, JAX

Table 13: Hyperparameters for the UUID experiments.

F GPT 4.1 Nano Prompt

We use the following prompt to GPT 4.1 Nano ([OpenAI, 2025](#)) to decide whether the language is correct in Figure 4. We replace <excerpt> with generator responses.

```
Classify the language of the following excerpt.  
Your answer must be the best choice of:  
English, Spanish, German, Italian, French, Not Natural Language.  
Output only your final choice with no explanation.  
Here is the excerpt: <excerpt>
```

G Wikipedia Paraphrase Prompt

We use the following prompt for our generator, where <article> is replaced with Wikipedia articles to paraphrase. Due to the prompt length limit (see Appendix E), the article is often truncated.

```
Help read the following article and then rephrase it in  
different terms. Remember to keep the meaning and every  
content of the article intact, including the title, year,  
etc. Here is the article:\n<article>
```