SOFTSRV: LEARN TO GENERATE TARGETED SYNTHETIC DATA.

Giulia DeSalvo, Jean-François Kagy, Lazaros Karydas, Afshin Rostamizadeh, Sanjiv Kumar
Google Research
New York, NY 10011, USA
{giuliad, jfkagy, lkary, rostami, sanjivk}@google.com

ABSTRACT

We present a novel framework, SoftSRV, that is used to generate targeted synthetic fine-tuning data for improving task-specific model performance. Given a sample from a target distribution, our proposed framework uses a data-driven loss minimization approach to steer a frozen large language model (LLM) to generate synthetic sequences that are similar to those from the target distribution. SoftSRV provides a practical improvement over common prompt engineering approaches that rely on human-engineered prompt-templates, which can be idiosyncratic, labor-intensive to craft, and may need to be specialized per domain. We empirically evaluate our method against standard baselines guiding a large LLM to generate synthetic data to fine-tune a smaller language model on three different domains (coding, math, reasoning). We perform these evaluations without any particular specialization of the framework to each domain, emphasizing the generality of our approach. We find that SoftSRV improves upon typical prompt engineering approaches, generating targeted data that leads to fine-tuned models with significantly better task-specific performance. In addition, SoftSRV-generated data better matches the target distribution according to the MAUVE similarity metric.

1 INTRODUCTION

In recent years, pre-trained large language models have proven to be effective in generating synthetic natural language training data (Gunasekar et al., 2023; Li et al., 2023; Eldan & Li, 2023; Mukherjee et al., 2023; Mitra et al., 2023; Abdin et al., 2024). This is particularly true when the synthetic data is used to pre-train or fine-tune smaller language models, enabling performances that rival models that are orders of magnitude larger (Liu et al., 2023). There are several motivations for generating and using synthetic training data; chief among them is the need to train models for domains where little natural high-quality text may be readily available or may be difficult to procure.

In order to generate synthetic text, a significant amount of human-driven prompt engineering is invested into developing prompts that steer the generating LLM into producing high-quality text from a targeted domain while also encouraging sufficient diversity. This point was nicely summed up by the authors of the open-source synthetic text repository Cosmopedia (Ben Allal et al., 2024), when recounting their work to recreate a large synthetic dataset similar to the one generated to train Phi 1.5 (Li et al., 2023): *"Heads up: If you are anticipating tales about deploying large-scale generation tasks across hundreds of H100 GPUs, in reality most of the time for Cosmopedia was spent on meticulous prompt engineering."* Furthermore, and especially in the case of generating fine-tuning data for targeted domains (e.g., coding, math, customer service), this manual process may need to be repeated and refined per-domain, or even per sub-domain (e.g., per coding language, math subject, service department). Apart from the human engineering cost, these manual prompting approaches do not directly optimize a data-driven objective. Rather they depend on human-in-the-loop style feedback for manually adjusting the prompt templates, resulting in approaches that lack robust mechanisms for aligning the LLM's generated data with the desired distribution. See further discussions of related work in Appendix B.

To address these issues, we propose an algorithmic framework, SoftSRV, that leverages trainable parameteric embeddings, rather than natural language prompts, to steer a pre-trained model towards

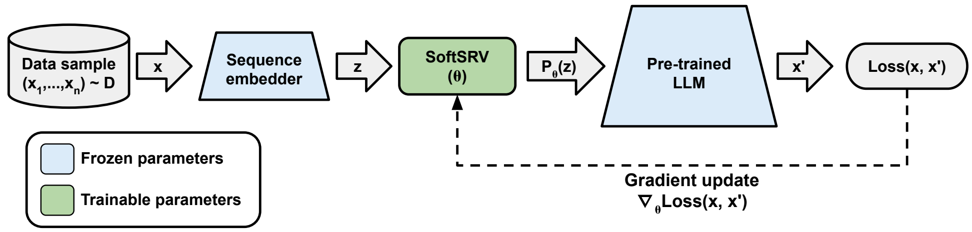

Figure 1: A diagram illustrating the training workflow of the SoftSRV framework. An example sequence x is embedded into a dense vector $\mathbf{z}$ via a (frozen) sequence encoder model. The SoftSRV model, parameterized by θ and conditioned on the embedding $\mathbf{z}$, produces a SoftSRV embedding $\mathbf{P}_\theta(\mathbf{z})$. This is then fed to a (frozen) pre-trained LLM, which produces a synthetic example x'. Similar to autoencoder-based training, the gradient of a next-word-prediction "reconstruction" loss is computed and used to update the SoftSRV parameters.

generating text that most resembles the target distribution. These parametric embeddings are trained by minimizing a data-driven loss function using an autoencoder-like compression and reconstruction procedure. We restrict SoftSRV to train on only a small number of parameters, thereby using a relatively limited amount of compute. Additionally, SoftSRV requires essentially no human-in-the-loop prompt engineering, enabling the process to be readily deployed across many domains.

Parameteric embeddings allow for more expressive inputs to an LLM compared to natural language prompts since an embedding is not restricted to correspond to a particular sequence of discrete natural language tokens. This intuitive observation is formalized in Petrov et al. (2024), which shows that in specific settings trained embeddings can induce an LMM to produce an exponential (in sequence length) number of text completions, while natural language prompts only allow for a linear number of completions. Although SoftSRV has some similarities to traditional prompt-tuning approaches, there are also several critical differences, which we discuss in detail in Appendix C.

The specific contributions presented in this work are:

- We introduce the SoftSRV framework and provide several instances of parameterized contextual embeddings that can be leveraged within the framework.
- We demonstrate that these embeddings can be successfully trained and used to generate targeted synthetic text for fine-tuning task-specific downstream models.
- Our empirical evaluations demonstrate that models fine-tuned on SoftSRV-generated text admit superior performance compared to those fine-tuned on data generated by baseline prompt engineering approaches or on limited amount of non-synthetic data. We show results on coding, math, and reasoning benchmarks, using both in-domain and out-of-domain tasks to study generalization of the synthetic data.
- We measure the similarity of the generated data to the target distribution using the MAUVE metric and observe that SoftSRV methods align most closely with the target distribution.

2 Proposed Approach

In this section, we introduce the general SoftSRV framework along with a few specific parameterizations. First, we start with some basic notation and terminology. Let $\{x_1, \ldots, x_n\}$ denote a sample of n text sequences, belonging to the set of all possible sequences S^m of a maximum length m, drawn according to a fixed but unknown distribution $\mathcal{D}$. Although we are not able to directly sample additional sequences from $\mathcal{D}$, our goal is to synthesize sequences that could have plausibly been drawn according to $\mathcal{D}$. We assume access to a (frozen) LLM, denoted $L : S^m \to S^m$, where we input and output sequences of equal fixed length m for notational simplicity and without loss of generality. Furthermore, we explicitly decompose the LLM, $L = H \circ I$, where $I : S^m \to \mathbb{R}^{d\times m}$ represents the initial embedding layer that embeds each token of the input sequence to a d-dimensional dense vector, and $H : \mathbb{R}^{d\times m} \to S^m$ represents the remainder of the language model that maps the embedded tokens to the output sequence. For SoftSRV, we rely solely on the frozen model H and we discard I entirely. As explained below, we will replace I by a parametrized family of embeddings.

The SoftSRV framework seeks to synthesize sequences similar to those drawn from $\mathcal{D}$ by training a dense embedding (or parameterized family of embeddings) $\mathbf{P} \in \mathbb{R}^{d \times t}$, with dimension $t < m$. We hereafter refer to $\mathbf{P}$ as the *SoftSRV embedding*. A successfully trained SoftSRV embedding, $\mathbf{P}$, should generate a sequence $x = H(\mathbf{P})$, via the frozen model H, that has a high likelihood of occurring under the distribution $\mathcal{D}$. More generally, we can sample several different sequences from a learned SoftSRV embedding, $x, x', x'' \ldots \sim H(\mathbf{P})$, by using randomized temperature-based decoding.

It is a recognized problem within the field of synthetic data generation that generated datasets often lack diversity (Li et al., 2023; Ben Allal et al., 2024). Although temperature sampling allows for some variability, it is important to inject more diversity in the generated sample. To this end, we further increase the variety of generated text by introducing a *contextual* SoftSRV embedding, $\mathbf{P}(\cdot) : \mathbb{R}^{d_e} \to \mathbb{R}^{d \times t}$. A contextual SoftSRV embedding can be conditioned with different context vectors $\mathbf{z} \in \mathbb{R}^{d_e}$, during training and generation, to induce greater variations in the SoftSRV embedding.

Before introducing specific parametrized families of SoftSRV embeddings, we describe the SoftSRV training procedure which is common throughout and also illustrated in Figure 1. We first let θ denote the trainable parameters of the contextual SoftSRV embedding $\mathbf{P}_\theta(\cdot)$. In addition to the sample of data $(x_1, \ldots, x_n)$ and frozen LLM (H), we assume access to a frozen sequence embedding function $E(\cdot) : S^m \to \mathbb{R}^{d_e}$ that will define our context vector $\mathbf{z}_i = E(x_i)$. During training, each training sequence is mapped to the context vector $\mathbf{z}_i = E(x_i)$ and used to generate a conditioned SoftSRV embedding $\mathbf{P}_\theta(\mathbf{z}_i)$, which is fed into the frozen LLM H to produce a new sequence $x'_i \sim H(\mathbf{P}_\theta(\mathbf{z}_i))$ using autoregressive next token generation. A standard causal (next-word) prediction loss, denoted $\ell(\cdot, \cdot)$, is backpropagated through the network up to the SoftSRV embedding layer $\mathbf{P}_\theta(\cdot)$, and an SGD-style update is applied to θ using the gradient $\nabla_\theta \ell(x_i, x'_i)$. This loss can be thought of as a "reconstruction" error and the entire pipeline is akin to an auto-encoder. Viewing the pipeline through this lens, it is apparent that the sequence embedder $E(\cdot)$ should be sufficiently "lossy" in order to avoid making the learning problem trivial. This lossiness can be enforced by restricting the dimension d_e of the embedding, for example.

Once the contextual SoftSRV embedding $\mathbf{P}_\theta(\cdot)$ has been trained, we can then generate synthetic data by passing $\mathbf{P}_\theta(\mathbf{z})$ to the frozen LLM as embedded input context for different choices of context vector $\mathbf{z}$. A natural choice is to sample embeddings $(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ derived from the data sample set $(x_1, \ldots, x_n)$.We now introduce a few specific SoftSRV parameterizations studied in this work.

SoftSRV Non-contextual Parameterization ($\mathrm{SS_{NP}}$). The simplest parameterization treats the dt entries of a SoftSRV embedding $\mathbf{P} \in \mathbb{R}^{d \times t}$, directly as trainable parameters, i.e., $\theta = \mathbf{P}$, resulting in the following objective: $\operatorname{argmin}_\theta \sum_{i=1}^{n} \ell(H(\mathbf{P}), x_i)$. This parameterization is an instance of a *non-contextual* SoftSRV embedding i.e., any context $\mathbf{z}$ is ignored.

SoftSRV Mixture Parameterization ($\mathrm{SS_{MPk}}$). Here, we train k "basis" SoftSRV embedding matrices and define the final SoftSRV embedding as a mixture of these bases. More precisely, in this variant the parameter set is $\theta = \{\mathbf{P}_1, \ldots, \mathbf{P}_k, \phi\}$, where $\mathbf{P}_i \in \mathbb{R}^{d \times t}$ are the basis SoftSRV embeddings, $\mathbf{P}_\theta(\mathbf{z}) = \sum_{i=1}^{k} w_i \mathbf{P}_i\,, \quad (w_1, \ldots, w_k) = W_\phi(\mathbf{z})$, and $W_\phi(\cdot) : \mathbb{R}^{d_e} \to \mathbb{R}^k$ is a learned softmax function with parameters $\phi \in \mathbb{R}^{d_w}$. The trained $\mathrm{SS_{MPk}}$ embedding is then the SGD solution to $\operatorname{argmin}_\theta \sum_{i=1}^{n} \ell(H(\mathbf{P}_\theta(E(x_i))), x_i)$.

SoftSRV MLP Concatenated Parameterization ($\mathrm{SS_{MC}}$)**.** Next, we consider a collection of t small MLPs, whose output is concatenated to generate the final SoftSRV embedding. Let $F_{\phi_i} : \mathbb{R}^{d_e} \to \mathbb{R}^t$ denote the ith MLP with parameters ϕ_i, and $\theta = \{\phi_1, \ldots, \phi_t\}$ denote the parameters for the collection of MLPs. Then, we define, $\mathbf{P}_\theta(\mathbf{z}) = \Big[F_{\phi_1}(\mathbf{z}), \ldots, F_{\phi_t}(\mathbf{z})\Big]$, and the trained $\mathrm{SS_{MC}}$ embedding is the SGD solution to $\operatorname{argmin}_\theta \sum_{i=1}^{n} \ell(H(\mathbf{P}_\theta(E(x_i))), x_i)$. This parameterization is the most expressive that we consider, in that each embedding column (which can be thought of as an "soft token") is computed using a distinct non-linear transformation of the context vector $\mathbf{z}$.

3 Empirical Evaluation

To empirically evaluate the SoftSRV framework, we consider a supervised fine-tuning setting where we fine-tune a small Gemma 2B model (Team et al., 2024) using synthetic data generated by a large foundational decoder-only language model. To generate the data, we only consider methods that directly prompt the larger model and/or train a relatively small number of parameters (as in the case

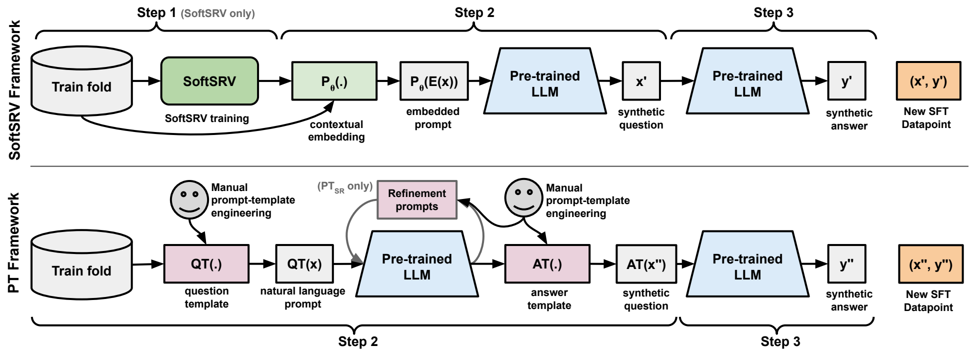

Figure 2: An illustration of the workflow for generating synthetic SFT data with SoftSRV (top panel) and natural language prompt templates (bottom panel); step numbering matches the discussion in Section 3. For SoftSRV, we use questions from the train set to (1) train a contextual embedding, (2) embed the same training sequences to serve as context vector to the trained embedding which is then fed to the LLM to generate synthetic questions, and (3) we generate answers by simply feeding the synthetic questions to an LLM. In the baseline prompt template framework, step (2) generates questions, using questions from the train set to fill the natural language template (optionally conduct rounds of refinement prompting), and (3) generates answers to these questions again using a template, but populated with the synthetic questions.

of SoftSRV). We do not assume the ability to conduct full fine-tuning of the large foundational model, as it is generally computationally prohibitive. To demonstrate the generality of the proposed approach, we generate fine-tuning data for several disparate domains (coding, mathematics, reasoning) using the same exact pipeline with no particular specialization to any of the domains. Specifically, we consider MBPP (Austin et al., 2021) for coding (metric: 3-shot pass@1), GSM8K (Cobbe et al., 2021) for math (metric: 5-shot pass@1), and BoolQ (Clark et al., 2019) for reasoning (metric: accuracy). See Appendix A.1 for more details on these domains.

We compare our methods against two prompt engineering approaches that involve manually creating natural language prompt templates that are then seeded with text from the desired target domain, usually taken from the training set. The first method, denoted simply as prompt template (PT), uses a domain-specific template to generate a question followed by another domain specific template to generate answers. The second approach, prompt template with self-refinement ($\mathrm{PT}_{\mathrm{SR}}$), similarly uses a template to generate questions but also iteratively conducts several rounds of self-critique to improve or accept the question (Madaan et al., 2023). For more details on these methods, please see Appendix A.2. We test against these approaches as they are currently the most effective, general, and widely used approaches for practical synthetic data generation, see related work in Appendix B.

The data generation and evaluation pipeline used for each approach proceeds in the following steps and is outlined in Figure 2. *Step 1. Train Parameters (SoftSRV only).* We train the contextual SoftSRV embeddings, θ, by minimizing a next-word-prediction loss against a frozen LLM. *Step 2. Generate Questions.* Both SoftSRV and template methods generate questions by using all training set examples and leveraging temperature sampling. We generate 50,000 questions for MBPP and GSM8K and 20,000 questions for BoolQ (see Appendix E for examples.) *Step 3. Generate Answers.* Both SoftSRV and prompt template methods leverage an off-the-shelf LLM for answer generation, but prompt template methods additionally employ domain-specific prompt templates. All methods undergo a decontamination process. *Step 4. Fine-tune and Evaluate Downstream Model.* We fine-tune the 2B Gemma 2 model on the generated question-answer pairs by each method and evaluate on the test set of each benchmark domain. The details of this pipeline are described in Appendix A.3.

Comparing our Parameterizations of SoftSRV. We start our empirical analysis by comparing different SoftSRV parameterizations to identify the highest performing configuration. Here, two salient questions are 1) Which parameterization most is effective and 2) whether it is essential to have a contextual SoftSRV embedding that leverages the context vector as opposed to a non-contextual SoftSRV embedding. In Table 1[left], we present the performance of Gemma 2B fine-tuned on synthetic datasets generated by $\mathrm{SS}_{\mathrm{NP}}$, $\mathrm{SS}_{\mathrm{MP2}}$ and $\mathrm{SS}_{\mathrm{MC}}$, respectively. These results show that the non-contextual method, $\mathrm{SS}_{\mathrm{NP}}$, generates synthetic fine-tuning data that results in lower performance

Table 1: On the left, we show downstream task performance of Gemma 2B models fine-tuned on the data generated by each of the three SoftSRV methods, evaluated at the 2K checkpoint. The number of non-synthetic examples used as seed data for the prompt templates and SoftSRV models is reported as N_r. On the right, we show out-of-domain task performance of Gemma 2B models fine-tuned on PT-generated data versus on $\mathrm{SS_{MC}}$-generated data, evaluated at the 2K checkpoint.

task	N_r	$\mathrm{SS_{NP}}$	$\mathrm{SS_{MP2}}$	$\mathrm{SS_{MC}}$
MBPP	384	0.296	0.312	0.372
GSM8K	7,473	0.413	0.401	0.463
BoolQ	9,427	0.806	0.825	0.831

task	PT	$\mathrm{SS_{MC}}$
Transcoder	0.141	0.369
MAWPS	0.583	0.547
TriviaQA	0.163	0.432

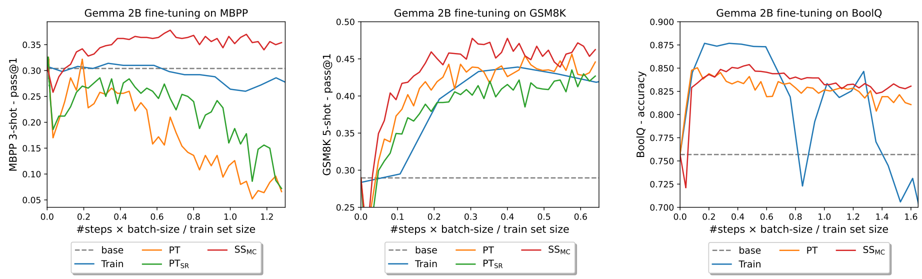

Figure 3: Gemma 2 (2B) fine-tuning curves for the synthetically generated datasets as well as the non-synthetic training set.

than the other (contextual) parameterizations, indicating the effectiveness of contextual SoftSRV embedding. Further, comparing the performance of the $\mathrm{SS_{MC}}$ and $\mathrm{SS_{MP2}}$ methods, we observe that the more expressive parameterization of $\mathrm{SS_{MC}}$ is generally beneficial. Consequently, we advocate for the use of the $\mathrm{SS_{MC}}$ method in preference to other parameterized families. We focus subsequent sections on comparisons with $\mathrm{SS_{MC}}$.

3.1 Comparing to Natural Language Templates

Here, we compare our best SoftSRV parameterization, $\mathrm{SS_{MC}}$, to the two natural language prompt template methods. We first report the performance of Gemma 2B fine-tuned on the synthetic datasets generated by PT, $\mathrm{PT_{SR}}$, and $\mathrm{SS_{MC}}$. Figures 3 plot the eval metrics for each dataset as a function of the number of fine-tuning steps times batch size normalized by training set size (essentially the training epoch modulo an additional constant factor due to sequence packing). We find that the model fine-tuned on $\mathrm{SS_{MC}}$-generated data outperformed the models fine-tuned on data generated by the two prompt template methods.

Comparing the prompt template methods, the PT method outperforms the $\mathrm{PT_{SR}}$ method on the GSM8K benchmark. For the MBPP benchmark, PT initially attains a similar performance as that of $\mathrm{PT_{SR}}$, but both methods start to degrade as a function of fine-tuning steps. This may be due to a lack of diversity in the generated text, given the small set of 384 MBPP training examples used to seed the prompt templates. We do not report the results of the $\mathrm{PT_{SR}}$ method for BoolQ as it appears to struggle to produce reasonable outputs, potentially due to the lengthy input passages.

The training datasets from our benchmark domains contain a relatively small number of examples (see column N_r of Table 1) and both prompt template methods and SoftSRV use this training data to generate tens of thousands of synthetic examples (see Step 2 in Section 3 for exact number). Yet, as shown by the results in this section, SoftSRV is more successful at leveraging this limited data. Consequently, we argue that SoftSRV is especially well suited to the data scarce setting.

Next, we investigate the impact of synthetic dataset size by fine-tuning the model on datasets of 20K, 50K, and 100K synthetic examples for the BoolQ benchmark. We find that $\mathrm{SS_{MC}}$ improves steadily with dataset size, performing 1.8 times better than PT. In contrast, the PT method only improves with 100K examples, achieving similar performance to $\mathrm{SS_{MC}}$ at 50K. Please see Appendix A.7 for result figures and further details.

Table 2: On the left, downstream task performance of Gemma 2B models fine-tuned on a mixture of Gemma 2 post-training (G2) with $\mathrm{SS_{MC}}$-generated data versus with the train data, evaluated at the 10K checkpoint. On the right, the MAUVE scores of synthetic datasets (and train set for comparison) computed with respect to the non-synthetic test sets.

task	G2 + Train	G2 + $\mathrm{SS_{MC}}$
MBPP	0.320	0.356
GSM8K	0.423	0.539
BoolQ	0.867	0.840

	Train	PT	$\mathrm{PT_{SR}}$	$\mathrm{SS_{MC}}$
MBPP	0.963	0.463	0.397	0.477
GSM8K	0.998	0.914	0.865	0.991
BoolQ	0.999	0.784	–	0.995

Finally, as an additional comparison, we test $\mathrm{SS_{MC}}$ generated data against the expertly curated GSM8K-specific synthetic dataset of Liu et al. (2023) (see detailed discussion in Appendix A.8). Even though differences in the synthetic generation setup introduce several confounding factors, it is nonetheless encouraging to see that the $\mathrm{SS_{MC}}$-generated data is comparable to this highly curated synthetic dataset.

Out-of-Domain Benchmarks: The primary focus of this work has been on generating task-specific synthetic data and, thus far, we have evaluated fine-tuned models on held-out test sets from the task-specific distribution. Here, we measure the performance of the fine-tuned models on different, but thematically related, benchmarks to understand the generalizability of each data generation method to out-of-domain tasks. Here, we consider the following additional benchmarks: TriviaQA (Joshi et al., 2017) with a 5-shot prompt (accuracy) for reading comprehension, Transcoder (Sun et al., 2023) with a 3-shot prompt (pass@1) for coding, and MAWPS (Koncel-Kedziorski et al., 2016) with 0-shot (accuracy) for math. Each of these metrics is tested on its corresponding model, e.g. for reasoning, we evaluate on TriviaQA the model fine-tuned on the BoolQ-based synthetic datasets, etc. Table 1[right] shows that the $\mathrm{SS_{MC}}$ method outperforms the PT template method on TriviaQA and Transcoder while exhibiting a closer performance on MAWPS.

Considering both in-domain results of Figures 3 and the above out-domain benchmarks, we find that, on the whole, the $\mathrm{SS_{MC}}$ method exhibits superior performance compared to the PT template method.

3.2 Comparing to the Original Training Data

Having identified $\mathrm{SS_{MC}}$ as the most promising synthetic data generation method from those we considered, we now proceed to evaluate its performance relative to the original (non-synthetic) training data from the benchmarks. Figures 3 show that the model fine-tuned on the $\mathrm{SS_{MC}}$ generated data outperforms the model fine-tuned on the non-synthetic train data for MBPP and GSM8K, indicating that, given enough of it, synthetic data can outperform even non-synthetic data. However, the same observation does not hold for BoolQ. The training set curve on BoolQ admits high variance, but it attains a higher accuracy overall. As discussed in Appendix A.1, we expect generating questions for the BoolQ dataset to be more difficult both due to the broad range of question topics and also due to the relatively long context-length needed to form a good question.

Next, instead of fine-tuning on targeted data only, we augment a standard Supervised Fine-Tuning (SFT) dataset, namely Gemma 2 post-training dataset, by mixing it with our datasets. In all experiments, we used an 80/20 mixture ratio (80% post-training data, 20% additional data – either $\mathrm{SS_{MC}}$ or original training data). Table 2[left] shows that models fine-tuned with the $\mathrm{SS_{MC}}$ data outperformed those fine-tuned with the original training data on MBPP and GSM8K, but underperformed on BoolQ, mirroring previous findings. Please see Appendix A.6 for further details.

3.3 Distribution Matching Analysis

Finally, we calculate the MAUVE scores, as described in Pillutla et al. (2021), to measure how similar the generated text is to the target distribution (see Appendix A.9 for details). Since higher scores indicate greater similarity, Table 2[right] shows the $\mathrm{SS_{MC}}$ generated data is significantly closer to the target distribution than prompt template methods, even approaching the similarity of the training data on some datasets. While not a direct indicator of downstream performance (see previous two subsections for such results), these findings provide additional support of SoftSRV as a high-fidelity text generation method.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

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

Loubna Ben Allal, Anton Lozhkov, and Daniel van Strien. Cosmopedia: how to create large-scale synthetic data for pre-training. `https://huggingface.co/blog/cosmopedia`, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Neural Information Processing Systems*, 2020.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

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

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Heting Gao, Kaizhi Qian, Junrui Ni, Chuang Gan, Mark A. Hasegawa-Johnson, Shiyu Chang, and Yang Zhang. Speech self-supervised learning using diffusion model synthetic data. In *International Conference on Machine Learning*, 2024.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. Language models can teach themselves to program better. In *International Conference on Learning Representations*, 2023.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Krishna Anumanchipalli, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Llm2llm: Boosting llms with novel iterative data enhancement. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Empirical Methods in Natural Language Processing*, 2021.

Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong Zhang, et al. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*, 2024a.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Yichuan Li, Xiyao Ma, Sixing Lu, Kyumin Lee, Xiaohu Liu, and Chenlei Guo. MEND: Meta demonstration distillation for efficient and effective in-context learning. In *International Conference on Learning Representations*, 2024b.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving> 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*, 2023.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *International Conference on Learning Representations*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Neural Information Processing Systems*, 2023.

S. Mahajan, T. Rahman, K. Yi, and L. Sigal. Prompting hard or hardly prompting: Prompt inversion for text-to-image diffusion models. In *Computer Vision and Pattern Recognition*, 2024.

Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, et al. Orca 2: Teaching small language models how to reason. *arXiv preprint arXiv:2311.11045*, 2023.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Neural Information Processing Systems*, 2023.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 2011.

Aleksandar Petrov, Philip Torr, and Adel Bibi. When do prompting and prefix-tuning work? a theory of capabilities and limitations. In *International Conference on Learning Representations*, 2024.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. MAUVE: Measuring the gap between neural text and human text using divergence frontiers. In *Neural Information Processing Systems*, 2021.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Huai hsin Chi, Nathanael Scharli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, 2023.

Qiushi Sun, Nuo Chen, Jianing Wang, Xiang Li, and Ming Gao. Transcoder: Towards unified transferable code representation learning inspired by human skills. *arXiv preprint arXiv:2306.07285*, 2023.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Longhui Yu, Weisen Jiang, Han Shi, YU Jincheng, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, 2024.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *International Conference on Computer Vision*, 2023.

A Empirical Evaluation Additional Details

This appendix provides supplementary details about our empirical evaluations.

A.1 Benchmark Task Discussion

The benchmarks below aim to cover a wide variety of tasks, each with varying degrees of complexity – both in terms of solving the task and in terms of generating synthetic data.

Code – MBPP (Austin et al., 2021). For the coding domain, we consider the "Mostly Basic Python Problems" (MBPP) benchmark, which consists of short Python programming exercises, e.g. "Write a python function to find the first repeated character in a given string" and answers written in Python code. The task is evaluated using a 3-shot prompt (i.e. a prompt pre-pended with three instructional examples) and pass@1 metric, measuring if a single top generated result is correct.

Math – GSM8K (Cobbe et al., 2021). For the math domain, we use the grade school math word problems from the GSM8K benchmark. This dataset contains only highly-curated word problems written by humans that are conceptually simple, but require multi-step reasoning. For this generative task we use a 5-shot prompt, again, measuring the pass@1 metric.

Reasoning – BoolQ (Clark et al., 2019). For general reasoning, we consider the binary question answering dataset from the BoolQ benchmark. The questions arise organically from anonymized Google search queries which can be answered as either 'true' or 'false'. Each question and answer is paired with a passage (average length of 108 tokens) that is extracted from a relevant Wikipedia page. This is evaluated as a scoring/classification task and accuracy is reported.

The MBBP task requires basic Python programming knowledge to solve, but the questions are generally short, follow a similar pattern and are concerned with a relatively narrow set of themes. The GSM8K task requires basic math and language comprehension skills, but generating the problem is arguably even harder than solving it. It requires generating a premise containing several numerical quantities and then a question that can be answered using the provided information in a non-trivial fashion. Nonetheless the premises are somewhat formulaic and thematically similar.

The BoolQ reasoning task, which requires general reading comprehension to solve, is perhaps the most difficult task to generate synthetic data for. Generating a problem requires writing a long (relative to MBPP and GSM8K) passage on an arbitrary topic that contains a collection of facts, but that does not necessarily stick to any formula or theme, and then generate a true/false question that can be answered directly by the passage. As we shall see in the empirical evaluation, the level of difficulty in generating a high-quality question can impact the relative quality and value of generated synthetic data.

A.2 Baselines

Typical prompt engineering approaches for synthetic text generation involve manually creating and populating natural language prompt templates with target domain text (typically from the training set). To give a simple illustrative example, a template could be:

```
Consider the following [article], write a summary of the
topic suitable for a high-school audience,
```

where the placeholder `[article]` would be replaced with example texts from training fold, producing several distinct prompts. In this study, we consider the following two natural language prompt template variants.

The first method which we called prompt template (PT) employs domain-specific templates for both question and answer generation (the detailed workflow is discussed in Section 3). In Appendix D, we provide the exact templates used by the PT method. To design these templates, we undertook several iterations of prompt engineering and reported the result of the best performing method. In particular, we found that prompting for a "diverse" set of questions was crucial (a comparison plot is presented in Appendix A.4).

The second method, prompt template with self-refinement ($\mathrm{PT_{SR}}$), also uses a template for question generation, then iteratively self-critiques and refines the questions over several rounds (Madaan et al., 2023). Again, critique and refinement prompts are shown in Appendix D.

A.3 Details on Empirical Evaluation Procedure

Below, we describe in details the steps of our empirical evaluation procedure.

Step 1. Train Parameters (SoftSRV only). For the SoftSRV methods, we first train its parameters with a frozen large decoder-only LM backbone using the questions found in the training fold of the benchmark dataset, which serves as our sample from the target distribution. We embed each question, $\mathbf{z}_i = E(x_i)$, and run an Adam optimizer to minimize the causal next-word-prediction loss, $\mathrm{argmin}_\theta \sum_{i=1}^{n} \ell(H(\mathbf{P}_\theta(\mathbf{z}_i), x_i)$, where $\mathbf{P}_\theta(\mathbf{z})$ is the contextual SoftSRV embedding and H is the frozen LLM (post input embedding layer). The sequence embeddings, $E(\cdot)$, are set to be the average of token embeddings computed by a small off-the-shelf decoder-only LM. This simple choice for $E(\cdot)$ is used to both limit the amount of additional computation, but also to ensure the sequence embedding is somewhat lossy in order to make the reconstruction task (i.e. minimizing $\ell(H(\mathbf{P}_\theta(E(x_i)), x_i)$ challenging). The simpler $\mathrm{SS_{NP}}$ method does not use this sequence embedding as it is not a contextual parameterization.

Since we are interested in a low-touch general framework, we avoid any domain-specific hyperparameter tuning for SoftSRV. Specifically, for all SoftSRV variants and all benchmarks, the length of the prompt t is fixed to be 128, the number of training steps was set to 20K, and the learning rate is fixed to $5\mathrm{e}{-6}$, which we found to be reasonable defaults. The $\mathrm{SS_{MC}}$ method uses MLPs with 3 feed forward layers and 128 hidden dimensions. For the $\mathrm{SS_{MPk}}$ variant, we primarily evaluate with $k = 2$ to limit to the total number of parameters, although a partial exploration for other values of k is presented in Appendix A.5.

Step 2. Generate Questions. The main application of SoftSRV is in this step, where we generate new questions for the target fine-tune task. Both the SoftSRV and the prompt template methods use all examples in the train data during this question generation phase.

Specifically, we pass in the questions from the training dataset, x_i, and produce a new sequence $x_i' \sim H(\mathbf{P}_\theta(E(x_i)))$ via temperature sampling (with default temp=1). For the $\mathrm{SS_{NP}}$ variant, no context vector and, thus, no training examples are used during generation. For all SoftSRV methods, no hard prompt template of any kind is used.

For the hard prompt baselines, we generate synthetic questions by querying the same backbone LLM using the relevant domain specific hard prompt template and questions taken from the benchmark training fold to populate the template. We conducted a search over temperature={1,2,4} and found a temperature of 2 to provide a balance of diversity and quality for these hard prompting methods.

For all methods, we generate 100K questions, repeating example questions from the training fold in a round-robin fashion. We then run a simple filtering, deduplication and subsampling pipeline to arrive at a target fine-tuning dataset size N_s. Concretely, from these 100K, we first filter exact duplicates. Then, to encourage a diverse subsample, again for all methods, we cluster the data and select examples from each cluster randomly in round-robin fashion. That is, using the scikit-learn library (Pedregosa et al., 2011), we apply MiniBatch k-means to vectorized data, which has been reduced in dimensionality using SVD. For all methods, we set the number of clusters for MiniBatch k-means to 700, reduced the dimensionality to 100 features and used sk.TfidfVectorizer for vectorization. Given the k-means clustering, we randomly select without replacement one point per cluster until N_s questions are chosen. We use N_s=50,000 for MBPP and GSM8K and N_s=20,000 for BoolQ.

Step 3. Generate Answers. After generating the questions, all methods essentially follow the same procedure to generate answers using an off-the-self LLM. The only difference being, in the case of SoftSRV, we pass the question directly to the LLM without any domain specific prompting to preserve its domain agnostic nature. In the case of the prompt template baselines, we use a domain specific template combined with the generated question to query the off-the-shelf LLM for an answer. For all methods, once we have full (questions, answer) fine-tuning examples generated, we run a decontamination process to remove any examples that may have been inadvertently leaked to the pretrained LLM, as is standard practice. Specifically, for all methods, we decontaminate the generated sequences against the respective benchmark's test set by removing any n-gram matches

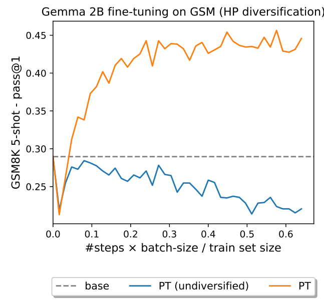

Figure 4: Performance of PT method with and without diversification, i.e. the phrase generate "10 different questions".

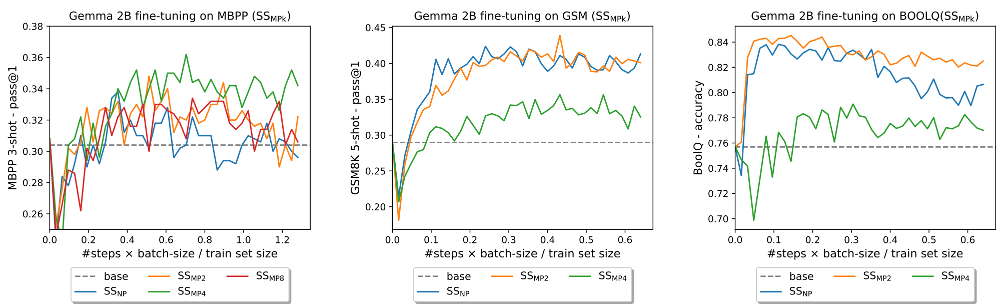

Figure 5: Comparison of $\mathrm{SS}_{\mathrm{MPk}}$ variants for different values of k. $\mathrm{SS}_{\mathrm{NP}}$ corresponds to $k = 1$.

where $n = 13$ as is common practice (Brown et al., 2020). Prior to calculating the matches, we eliminate all punctuation and numerical characters. We found that the contamination of the generated sequences to the test set is minimal with less than 0.1% for GSM8K and MBPP and around 1% for BoolQ.

Step 4. Fine-tune and Evaluate Downstream Model. Finally, for all methods, we use the generated (question, answer) pairs to fine-tune the target 2B Gemma 2 model. We use a batch-size of 16 with sequence length 8192 and with a learning rate with linear warmup from 0 to 1e-6 over 100 steps, followed by a cosine annealing schedule.

We evaluate the performance of these fine-tuned models on the test fold of the respective benchmarks.

A.4 Diversification of the PT method

To arrive at the the PT method presented in the main body of the paper, we conducted multiple iterations of prompt engineering and template refinements. In particular, we found that asking the model to generate "10 different questions" per example question and using a higher decoding temperature was critical. We demonstrate this effect on the GSM8K benchmark in Figure 4 which compares the performance of a model fine-tuned on data generated by PT method to that of the model fine-tuned on its undiversified counterpart where the question template asks to generate one question per given example question and the default decoding temperature is used. We found similar results for the other datasets. Despite the relative small change in the settings, the difference in model performance is significant, demonstrating some of the idiosyncratic nature of prompt template approaches. We provide the template for the undiversified PT in Appendix D.

A.5 Mixture of Prompts with Various Values of k

Here, we conduct a exploration to measure the effect of changing the number of basis embedding matrices, k, of the $\mathrm{SS}_{\mathrm{MPk}}$ method.

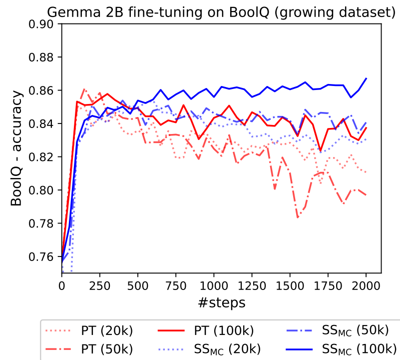

Figure 6: We compare the BoolQ performance of Gemma 2B fine-tuned on data generated by PT and $\mathrm{SS}_{\mathrm{NP}}$ as the number of generated examples increases.

Figure 5 shows the comparison in performance on various benchmarks across different values of k. We see an inherent trade-off as we increase k, which increases the capacity of the contextual embedding but also then require training more parameters. For MBPP we see that the value $k = 4$ appears to be optimal, while for GSM8K and BOOLQ we see performance peaks at $k = 2$. We fix $k = 2$ throughout the evaluations in the main paper as a good general choice across different tasks.

A.6 Supplementing a post-training dataset

In the previous evaluations, we considered a setting where we fine-tune using either the synthetically generated data or the original training data. Here, we investigate augmenting an existing general SFT dataset to evaluate the effectiveness of our datasets when combined with a mixture of data sources. Specifically, we compare the performance of a 2B Gemma 2 model fine-tuned on the Gemma 2 post-training dataset Team et al. (2024) augmented with the synthetic $\mathrm{SS}_{\mathrm{MC}}$ data versus augmented with the original training data. For all experiments, we fixed a priori the mixture source ratio to 80% for the post-training dataset and 20% for the added set (i.e. either $\mathrm{SS}_{\mathrm{MC}}$ data or training data). All other parameters remain consistent with previous sections except here the models are trained for 10K steps as the amount of data is greater.

Table 2[left] shows that on MBPP and GSM8K, the model fine-tuned on the mixture of Gemma 2 post-training data with $\mathrm{SS}_{\mathrm{MC}}$ generated data outperforms the model trained on the mixture of the post-training with the train data. On BoolQ, similar to our previous findings (see first paragraph of Section 3.2), the $\mathrm{SS}_{\mathrm{MC}}$-based model underperforms compared to the model using the train data.

A.7 Data Scaling

Here, we analyze the effects of varying the number of synthetic examples generated when fine-tuning only on synthetic examples. Specifically, we increase the number of generated examples from 20K, 50K, to 100K, testing both the $\mathrm{SS}_{\mathrm{MC}}$ and PT approaches for the BoolQ benchmark, given this benchmark appears the most challenging in terms of generating effective synthetic questions. Figure 6 shows that the performance of the model fine-tuned on PT stagnates between 20K and 50K synthetic examples, and only shows improvement when the number of generated examples reaches 100K. In contrast, the model fine-tuned on $\mathrm{SS}_{\mathrm{MC}}$ steadily improves as the generated dataset size increases. Going from 20K to 100K examples, $\mathrm{SS}_{\mathrm{MC}}$ performance increases at a 1.8 times faster rate relative to PT with respect to their max values. In particular, the PT method with 100K generated examples admits a comparable performance to that of the $\mathrm{SS}_{\mathrm{MC}}$ method with 50K generated examples.

A.8 Comparison with Curated Prompt Template Generated Dataset (TinyGSM)

Here, we compare against a high quality synthetic dataset, TinyGSM, within our fine-tuning evaluation framework. TinyGSM was expertly curated by Liu et al. (2023) for GSM8K-PAL, which is a program aided language model (PAL) variant of GSM8K that asks for questions to be answered in the form of

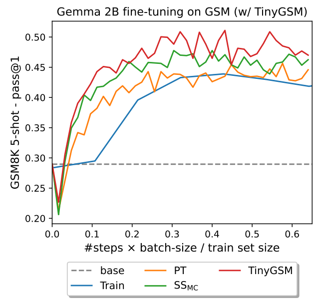

Figure 7: For the GSM8K benchmark, we show the Gemma 2B performance after fine-tuning on the $\mathrm{SS}_{\mathrm{MC}}$ and PT generated datasets against a sample of the same size from the curated TinyGSM dataset.

Python functions. This has the advantage of enabling verification of the answer in a programmatic fashion. Liu et al. (2023) use GPT-3.5-turbo with prompt templates seeded with training questions from the original GSM8K dataset and from the GSM-IC dataset, which is a dataset crafted to incorporate irrelevant context in order to bolster model robustness (Shi et al., 2023). They use two types of prompts: the first asks to generate both questions and answers while the second requires two calls to the LLM to first generate a question and then an answer. Leveraging the fact that the solutions of the math word problems are written in Python, they then filter out any data that contains code that is not executable by a Python interpreter. They additionally filter out questions that do not contain numbers as this indicates flawed math problems.

In order to evaluate the TinyGSM generated question in our setting, we randomly sample 100K questions from the publicly available TinyGSM dataset, then further subsample down to 50K using the same post-processing pipeline used by all other methods in our comparison. Finally, we generate answers, fine-tune and evaluate in the same fashion as the other prompt template baselines. Figure 7 shows that a model fine-tuned on the $\mathrm{SS}_{\mathrm{MC}}$ dataset attains a performance close to that when fine-tuned on the sample from the $\mathrm{TinyGSM}$ dataset while the PT method lags behind both. TinyGSM performing closely to $\mathrm{SS}_{\mathrm{MC}}$ is encouraging given that the TinyGSM dataset is highly curated and tailored specifically for the GSM8K benchmark.

A.9 Distribution Matching Analysis

In addition to our evaluations of the fine-tuned model's downstream performance, we investigate the capacity of SoftSRV and the prompt template baselines to generate data samples that can match a target distribution of text. To assess the proximity between the generated and target distributions, we compute MAUVE scores as in Pillutla et al. (2021), which can be viewed as a scalar summary of the divergence between the textual output of a generative model and a reference distribution. The MAUVE score is able to simultaneously measure both the model's ability to avoid generating text outside the support of the target distribution (Type I error) and the ability to generate text with a large coverage of the target distribution support (Type II error). The method essentially computes a quantized distribution for the generated and target distribution and measures their KL divergence, producing a normalized score between 0 and 1, where 1 indicates the two distributions are maximally similar (for further details se below Appendix A.9.1).

In Table 2[right], we report the MAUVE scores for the template prompt methods and for $\mathrm{SS}_{\mathrm{MC}}$ on the MBPP, GSM8K and BoolQ domains, measuring the distance to questions in the test fold. In all cases, we find that $\mathrm{SS}_{\mathrm{MC}}$ can synthesize text that is closer to a sample from the target distribution than the prompt template approaches. Notably, the $\mathrm{SS}_{\mathrm{MC}}$ method achieves a high MAUVE score that is even close to that of the train set on GSM8K and BoolQ, where it appears the additional flexibility afforded by its parameterization allows a very high-fidelity match to the target distribution. While the prompt template-based datasets generally have lower MAUVE scores than $\mathrm{SS}_{\mathrm{MC}}$, the PT variant is able to achieve a relatively high score on GSM8K. In the case of the MBPP dataset, we measure very low similarity scores for $\mathrm{PT}_{\mathrm{SR}}$; we conjecture this may be due to the several rounds of rewriting, which results in straying further from the original seed question.

Even though the MAUVE score is not a direct indicator of downstream fine-tuning performance (see our experiments in Section 3.1 for performance comparisons), it is an additional signal to provide further support for SoftSRV as a high-fidelity approach for text generation across domains.

A.9.1 MAUVE SCORE COMPUTATION

We let $\mathcal{G}$ and $\mathcal{D}$ denote the generated and target/reference distributions, respectively, and compute the MAUVE scores for each synthetic dataset as follows. Using an embedding model, a vector representation is computed for each sequence in the synthetic and reference sets. These embeddings are then projected into a discrete set using k-means clustering, and a divergence curve is traced between the cluster distributions of $\mathcal{G}$ and $\mathcal{D}$, see Equation 1 in Pillutla et al. (2021). The MAUVE has value between 0 and 1 and corresponds to the area under the divergence curve, where higher scores are indicative of a closer match between $\mathcal{G}$ and $\mathcal{D}$. The same small LM serving as the context embedder E for SoftSRV in Appendix A.3 is used here to compute per-token representations, which is then averaged to produce the sequence-level embedding. We use $k = 32$ clusters for all domains as we found that $k = 16$ or $k = 64$ yields similar qualitative results.

B RELATED WORK ON SYNTHETIC DATA GENERATION

As mentioned in the introduction, there is a significant recent body of work demonstrating the effectiveness of using a LLM to generate synthetic training data for a smaller model.

In terms for generating pre-training data, the collection of "Textbooks Are All You Need" white-papers outlines the process of training the Phi series of small LMs, using carefully curated prompt templates and seed data sources, and shows the large boost in quality that synthetic data can provide (Gunasekar et al., 2023; Li et al., 2023; Abdin et al., 2024). The Cosmopedia project (Ben Allal et al., 2024) conducts a similar study, while open-sourcing the prompts and generated data. Li et al. (2024a) construct a pretraining set almost "from scratch" using a diverse set of prompt templates by using taxonomies of fields/sub-fields/disciplines within an area of expertise. Apart from focusing on pre-training rather than fine-tuning, these works require a non-trivial amount of human effort for building and/or curating prompt templates for the generating LLM, which our effort seeks to minimize.

Mukherjee et al. (2023); Mitra et al. (2023) focus on building synthetic data for better instruction tuning. In particular, they start with the FLAN-V2 instruction tuning dataset and ask a LLM to expand on the terse responses in different verbose styles (specified by so-called "system instructions") to introduce variation in presentation and approach. Although, shown to be quite effective across a broad array of reasoning tasks, in our setting we wish to generate data focused on a specific target task, likely requiring us to curate a set of bespoke "system instructions" for each task.

Several works build fine-tuning data for specific domains, such as coding (Haluptzok et al., 2023; Luo et al., 2024) or mathematics (Yu et al., 2024). Although quite successful, these approaches leverage specific qualities of the target domain, for example, using a code interpreter to check correctness of generated code or using the fact that math problems contain numerical quantities that can be masked or manipulated to create variations of the original question.

Finally, Lee et al. (2024) recently proposed an adaptive procedure where a LLM is used to generate targeted fine-tuning data for a small model based on examples that the small model has made mistakes on. The LLM is prompted to rewrite variants of these questions using specialized per-domain prompt templates. Extending SoftSRV to an adaptive setting is a potentially promising future line of work.

C RELATED WORK ON PROMPT-TUNING & NON-TEXT DATA MODALITIES

Prompt-tuning is primarily known for its use as a parameter efficient fine-tuning method (Lester et al., 2021; Li & Liang, 2021). It has also been used as a framework to learn or compress in-context instructions (Mu et al., 2023). Recently, Li et al. (2024b) proposed training a secondary model to compress in-context instructions into soft prompt that is then prepended to the task prompt templates.

Although SoftSRV and existing prompt-tuning methods (Lester et al., 2021; Li & Liang, 2021) both train embeddings, we outline several crucial differences between the two approaches. (1) The application: prompt-tuning is generally used to prepend a prompt embedding to an existing natural language prompt that is then fed to an LLM, with the goal of providing a higher quality response; in this work, our goal is to generate synthetic training data by feeding an LLM a trained embedding alone, i.e. there is no natural notion of an input/output pair in this application. (2) Contextual embedding: in standard prompt-tuning, it typically suffices to train a single prompt embedding that is prepended to different input prompts; in our setting, since we feed only the prompt embedding (and no natural language prompt) to the LLM we must do more than train a static embedding if we wish to induce significant variation. To that end, we introduce and train parameterized *contextual* embeddings. (3) The optimization procedure: in standard prompt-tuning the input/output sequence pairs (taken naturally from the application) can be used to train the prompt embedding; in the case of SoftSRV we train using an autoencoder-like compression and reconstruction scheme.

While our study has been focused on generating synthetic text, there have been similar efforts in other modalities. For example, the ControlNet approach of Zhang et al. (2023) trains a diffusion model to produce images conditioned on contextual input, for example, image edges or 3D pose skeletons. Similarly, Gao et al. (2024) trains a diffusion-based speech model to condition on a "simple speech representation" embedding to guide the generation of new synthetic speech data. Finally, in the case of text-to-image generators, there has been a significant amount of work in solving the "inverse" problem of mapping from an image back to a prompt, so that one can more predictably generate synthetic images in certain styles (see Mahajan et al. (2024) and many references therein).

D Natural Language Prompt Templates

Below, we report the templates used for prompt template baselines for each benchmark dataset. Figure 8 provides the question templates while Figure 9 shows the answer templates. Figure 10 provides the template for the undiversified PT method, described in Appendix A.4, for the GSM8K benchmark. Figure 11 reports the critique and refine templates for $\mathrm{PT}_{\mathrm{SR}}$.

E Generated Text Examples

In Figure 12, we provide examples of synthetic questions for GSM8K (math questions), MBPP (Python questions) and BoolQ (passage and question pairs) generated using the $\mathrm{SS}_{\mathrm{MC}}$ method. Notice that the questions generated for BoolQ require longer range dependencies and may be more complex to generate and answer. Finally, for comparison, in Figure 13, we show examples from the train split of each dataset. These train set questions appear similar in style to the questions generated by the $\mathrm{SS}_{\mathrm{MC}}$ method.

MBPP Question Template:

```
Consider the following python question:
      [insert example question]
Now generate 10 different questions that require writing a Python
function similar to the example above.  Make sure each question is
different and sufficiently rephrased.  Please make sure you generate
questions, and not answers.  Please make sure each question you generate
has a well-defined answer.
Question 1:
```

GSM8K Question Template:

```
Consider the following grade-school math problem:
      [insert example question]
Now generate 10 different questions that require solving a
grade-school math problem similar to the example above.  Make sure
each question is different and sufficiently rephrased.  Please make
sure you generate questions, and not answers.  Please make sure each
question you generate has a well-defined answer.
Question 1:
```

BoolQ Question Template:

```
Consider the following passage and question:
      [insert example question]
Now generate 10 different passages and questions similar to the example
above.  Please make sure each question you generate has a boolean
answer that can be answered by the passage.  Make sure each passage
and question is different and sufficiently rephrased.  Please make sure
you generate passages and questions, and not answers.
Passage and Question 1:
```

Figure 8: Question template for MBPP, GSM8K and BoolQ benchmarks for the PT method.

MBPP Answer Template:

```
Please answer the following python question:
      [insert example question]
Please generate your answer as a Python function.  The docstring of
the function should contain the above question as-is, without any
modification.  Please make sure that your function is valid Python code
that compiles.  Please try your best to correctly answer the question.
Answer:
```

GSM8K Answer Template:

```
Please answer the following question that tests reasoning:
      [insert example question]
Answer:
```

BoolQ Answer Template:

```
Please answer the following question based on the passage.  Your answer
should be either True or False.  Do not provide any other justification.
      [insert example question]
Answer:
```

Figure 9: Answer template for MBPP, GSM8K and BoolQ benchmarks for the PT and $\mathrm{PT}_{\mathrm{SR}}$ method.

GSM8K Question Template for undiversified PT:

```
Please generate a question that requires solving a grade-school math
problem. Here is an example of such a question:
      [insert example question]
Now generate a new question. Please make sure your question is not
too similar to the example above. Please make sure you generate
a question, and not an answer. Please make sure the question you
generate has a well-defined answer.
```

Figure 10: Question template for the undiversified PT method for the GSM8K benchmark.

Critique Template:

```
Please provide actionable feedback on the clarity, difficulty, and
originality of the following {Python question, grade school math
problem, passage/question problem}:
      [insert question]
```

MBPP Refine Template:

```
Read the following Python question and the critique, and write a new
Python question based on the critique:
Question:
      [insert question]
Critique:
      [insert critique]
If the critique is strongly positive, say 'Stop'. Otherwise, write a
new Python question in a single sentence starting with 'Write a Python
function' based on the critique. Do not ask for docstring or test
cases.
```

GSM8K Refine Template:

```
Read the following grade-school math problem and the critique, and write
a new grade-school math problem based on the critique:
Question:
      [insert question]
Critique:
      [insert critique]
If the critique is strongly positive, say 'Stop'. Otherwise, write a
new grade-school math problem based on the critique. Write the question
only, do not include the answer.
```

BoolQ Refine Template:

```
Read the following passage/question problem and the critique, and write
a new passage/question problem based on the critique:
Question:
      [insert question/passage]
Critique:
      [insert critique]
If the critique is strongly positive, say 'Stop'. Otherwise, write a
new passage/question problem based on the critique. Write the passage
and question only, do not include the answer.
```

Figure 11: Refine template for MBPP, GSM8K and BoolQ benchmarks for the $\mathrm{PT}_{\mathrm{SR}}$ method.

GSM8K Generated Text Examples

```
Julian is twice as old as his little brother, Eli. In
12 years, Julian will be 37. How old is Eli today?
```

```
Sarah used 1/3 of her grass cutting saw’s fuel in
cutting the grass in front of her house and used 2/5
of the remaining fuel to cut the grass in her back
yard. If she started with 30 liters of fuel, how much
fuel did she use in cutting the front yard?
```

```
A boy is jogging at 4 m/s. How long will he take to
cover a distance of 5 km?
```

MBPP Generated Text Examples

```
Write a python function to find the average of three
given numbers.
```

```
Given a sequence of characters, create a new sequence
such that every character is repeated k times.
```

```
Write a function that takes in two numbers and returns
the sum of the two numbers.
```

BoolQ Generated Text Examples

```
PASSAGE: There are no known cases of anyone catching
a common cold by exposure to the cold weather. The
most likely cause of a common cold is a virus. The
symptoms of the common cold can include a runny nose,
sore throat, cough, sneezing, nasal congestion, and
watery eyes. A common cold is considered to be mild
when only mild symptoms are present. Severe symptoms
include sinus pain and fever.
QUESTION: can the common cold be caused by cold
weather
```

```
PASSAGE: Land Before Time V: The Mysterious Island
(1997) is an American animated direct-to-video film.
It is the fifth film in The Land Before Time series
and the first film in the series to be animated in
Canada and also the first film in the series to be
created by Universal Cartoon Studios.
QUESTION: is the land before time 5 a sequel
```

```
PASSAGE: The semilunar valves of the heart (or
sometimes called the sigmoid valves) are the valves
that prevent backflow from the aorta and pulmonary
artery. They are situated at the beginning of these
arteries. The pulmonary semilunar valve is on the
right side of the heart, while the aortic semilunar
valve is on the left. They both have three leaflets,
each of which is attached to the arterial wall by the
corpora fibrosa. They are attached to the artery in
such a manner that they cannot fold over to block the
flow of blood.
QUESTION: are the aortic and pulmonary valves
semilunar valves
```

Figure 12: Examples of generated synthetic questions generated by $\mathrm{SS}_{\mathrm{MC}}$.

GSM8K Example Questions From Train Set

```
A choir was singing a song that involved 30 singers.
In the first verse, only half of them sang. In the
second verse, a third of the remaining singers joined
in. How many people joined in the final third verse
that the whole choir sang together?
```

```
Kyle is 5 years older than Julian. Julian is 20 years
younger than Frederick. Frederick is 2 times older
than Tyson. If Tyson is 20, how old is Kyle?
```

```
Mr. Williams bought 10 gallons of juice for a party.
Each gallon has 10 cups. At the party, 5 cups of juice
were left. How many cups of juice were drunk?
```

MBPP Example Questions From Train Set

```
Write a function to move all the numbers in it to the
given string.
```

```
Write a function to find the largest subset where each
pair is divisible.
```

```
Write a function to increment the numeric values in
the given strings by k.
```

BoolQ Example Questions From Train Set

```
PASSAGE: Since being fixed on the fourth Thursday in
November by law in 1941, the holiday in the United
States can occur on any date from November 22 to 28.
When it falls on November 22 or 23, it is not the
last Thursday, but the penultimate Thursday in
November. Regardless, it is the Thursday preceding
the last Saturday of November.
QUESTION: is thanksgiving the last thursday of
november every year
```

```
PASSAGE: The Rocky Mountains, also known as the
Rockies, are a major mountain range in western North
America. The Rocky Mountains stretch more than 3,000
miles (4,800 km) from the northernmost part of British
Columbia, in western Canada, to New Mexico, in the
Southwestern United States. Within the North American
Cordillera, the Rockies are somewhat distinct from the
Pacific Coast Ranges, Cascade Range, and the Sierra
Nevada, which all lie further to the west.
QUESTION: is the sierra nevada part of the rocky
mountains
```

```
PASSAGE: The first player to get rid of their last
card (``going out'') wins the hand and scores points
for the cards held by the other players. Number cards
count their face value, all action cards count 20, and
Wild and Wild Draw Four cards count 50. If a Draw Two
or Wild Draw Four card is played to go out, the next
player in sequence must draw the appropriate number of
cards before the score is tallied.
QUESTION: can you finish a uno game on a wild card
```

Figure 13: Examples of questions from the train sets.