COOKBOOK: A FRAMEWORK FOR IMPROVING LLM GENERATIVE ABILITIES VIA PROGRAMMATIC DATA GENERATING TEMPLATES

Avanika Narayan*, Mayee Chen*, Kush Bhatia, Christopher Ré
Department of Computer Science
Stanford University
Stanford, CA
{avanika, mfchen, kush, chrismre }@cs.stanford.edu

ABSTRACT

The prevailing way to improve the generative capabilities of large language models (LLMs) is via fine-tuning on instruction datasets (e.g., OpenHermes, OpenOrca). These datasets have several limitations: they are costly to curate, might violate user privacy agreements or terms of service of LLM providers, and are unclear in terms of the task skills (e.g., "rules") the model learns from their instruction samples. In this work, we introduce COOKBOOK, a framework that uses data generating templates—simple Python functions over random tokens—to produce programmatic training data which improves LLM task performance. By defining simple data generating functions that encode the underlying "rules" of generative tasks, our framework can efficiently produce data without privacy concerns or the need for extensive curation. In the single-task setting, we show that COOKBOOK is effective across a wide range of tasks from document QA to entity disambiguation, providing performance gains of up to 60.1 accuracy points. We extend COOKBOOK to a multi-task setting, proposing an algorithm to optimally mix data from various templates, thereby addressing the challenge of improving LLMs across a broad range of tasks. On the standard multi-task GPT4ALL evaluation suite, we find that, averaged across tasks, MISTRAL-7B fine-tuned using COOKBOOK is the best 7B parameter instruction-tuned model, and, on an individual task level, is the best performing model on 3/8 tasks. Finally, to analyze COOKBOOK, we introduce the template alignment statistic as a novel metric for understanding how training on template-generated data enhances model performance by adhering to the task-specific rules encoded in the templates.

1 INTRODUCTION

Fine-tuning large language models (LLMs) on instruction datasets (Lian et al., 2023a; Longpre et al., 2023; Achiam et al., 2023; "Teknium", 2023) can significantly improve LLM generative capabilities. The success of these instruction datasets is typically attributed to their size, diversity, and degree of human alignment (Wang et al., 2022a). However, it is unclear what is being learned from these instruction datasets, specifically, what task "rules" the model learns from such data—i.e., does training a model on document-based question answering (QA) examples truly teach the model how to localize information in a passage? Additionally, due to their size and diversity, these datasets can be time-consuming and expensive to curate. Recent alternatives such as user chat logs (i.e., shareGPT) and LLM-generated data (Lian et al., 2023a; "Teknium", 2023) are cheaper but may violate user privacy and the terms of service of LLM providers. In short, we lack a dataset curation method that allows us to teach specific task rules, is inexpensive, and does not pose privacy or legal issues.

As a first step in addressing these challenges, we ask whether LLM capabilities can be improved by using programmatic data: data generated by simple functions which encode the underlying rules of a generative task. Consider the toy task of teaching the model how to *copy sentences*; for an input sentence "The day is sunny.", the output is "The day is sunny." The task's underlying rule, which is that the input and output should be equal, can easily be converted into a data generating function by

sampling identical strings for input and output, such as {input: "aabc", output: "aabc"}. In contrast to instruction-tuning datasets, this programmatic data can be generated cheaply and does not pose any privacy or legal concerns. Notably, there is an explicit rule of how inputs and outputs are generated that this data aims to teach the model, unlike in existing instruction datasets.

However, it is unclear how rules for complex open-ended generation tasks can be captured by data generating functions. For example, document QA has many rules ranging from localization and retrieval to contextual analysis—none of which clearly translate into a data generating function. Given a set of natural language (NL) tasks, we wish to (1) know how to construct data generating functions for each task rule, (2) combine rules to improve performance across several tasks at once, and (3) understand if the taught rules will generalize to the respective downstream NL tasks.

We propose COOKBOOK, a framework that uses a data generating *template* (the "recipes")—a simple Python function—to produce programmatic training data for improving LLM performance on a given task. These templates have two key properties: 1) they invoke task-specific data generating functions, which specify how inputs and outputs are generated in order to approximate the given task's rule, and 2) they operate over a random token space, which not only avoids privacy and legal concerns but also facilitates ease of data specification. Importantly, while the data generating functions are task-specific, they are composed using simple list operations—sampling, span selection, shuffling, replacement and concatenation. For instance, consider the template for document QA (Figure 1), wherein one must answer a question based on a document. One rule for solving this task is to localize the question in the document and retrieve the correct answer from nearby. Our template first constructs the document as a random sequence uniformly sampled over the token vocabulary. In order to instantiate the *localize and retrieve* rule, our template samples the question as a small span of the document token sequence and defines the answer as the span of tokens surrounding the question.

In the second part of our framework, we extend our template-based approach to the multi-task setting, where we are interested in fine-tuning the model across data generated from several task templates to improve many generative capabilities simultaneously—similar to what instruction datasets aim to achieve with sample diversity. To construct a dataset using multiple templates, we need to determine the proportions of their respective samples in the data mixture. One can set these proportions manually through trial-and-error, but this can be resource-intensive. To automate this process, we propose an aggregation algorithm which takes as input the downstream accuracy scores of models fine-tuned on samples from individual templates and sets the proportions as the softmax of these scores. Notably, these proportions are theoretically optimal under the assumption that accuracies are linear in proportions, which we empirically validate. This approach, however, requires labeled evaluation data to produce accuracy scores. We generalize the algorithm to the unlabeled data setting by using weak supervision (WS) methods to estimate the accuracy of each template (Ratner et al., 2019).

Given these templates, it is important to understand how the data generated by them impacts model performance on downstream tasks. In the third part of our framework, we introduce *template alignment scorers* that score how closely a given NL task sample's input and output follow the template's data generating function. For example, in Figure 1 document QA's alignment scorer measures on an NL sample the fraction of the question's words that are within a certain word radius of the answer in the document, testing how well the *localize and retrieve* rule applies to NL data. These scorers help us study if training on template-generated data improves performance by impacting samples where we expect the template's rule to be most applicable—samples with a high alignment score. Using alignment scorers, we then propose a new metric, called the *template alignment statistic*, to measure the extent to which the model's improvement is due to the model better following the template's rule.

Empirically, COOKBOOK enables us to generate programmatic data that improves LLM performance on generative tasks, and to our knowledge, we are the first to do so. Across 5 tasks (document QA, retrieval, commonsense QA, entity matching, and entity disambiguation) we find that models fine-tuned on data generated by our templates can improve performance on the corresponding NL tasks by up to 60.1 accuracy points. We analyze our approach by using the template alignment statistic to examine how performance relates to learning the intended rules and by studying individual components of templates (data generating functions, random tokens). We find that the structure imposed by the data generating function is critical for improving performance and that training on

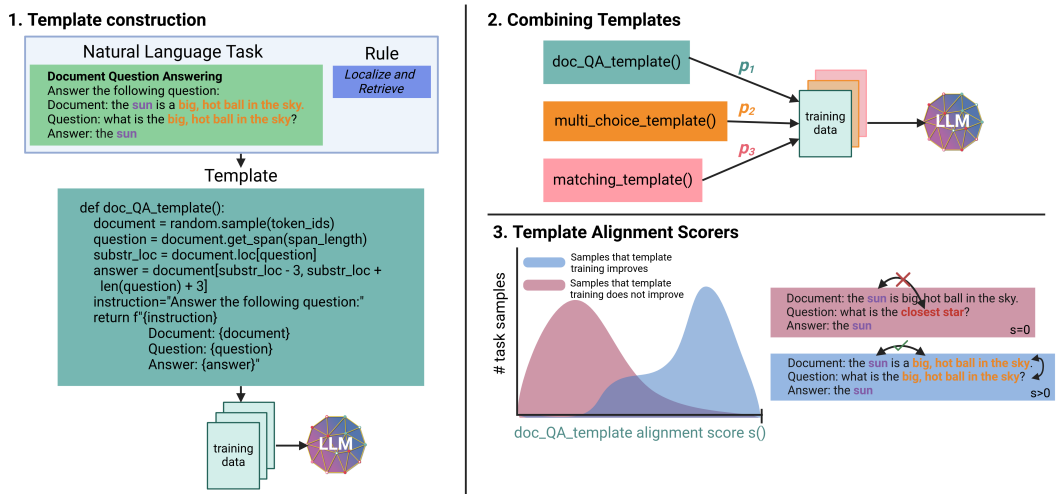

Figure 1: **COOKBOOK.** (1) Templates are constructed for a task by writing a data generating function representative of a task rule. (2) Templates can be combined to improve multiple capabilities. (3) Template alignment scorers help measure the degree to which the rule learned by the template improves LLM performance.

random tokens helps avoid reasoning shortcuts present in NL fine-tuning, given that the model already has sufficient NL understanding.

Finally, in the multi-task setting, we evaluate our data mixing algorithm over COOKBOOK templates, comparing performance of this fine-tuned model with models fine-tuned on existing instruction datasets (e.g., OpenOrca, Open-Hermes, Capybara) using the standard multi-task GPT4ALL evaluation suite. Averaged across tasks, we find that a model fine-tuned using the mixture of templates obtained from our algorithm is the best 7B parameter instruction-tuned model, and, on an individual task level, is the best performing model on 3 out of 8 tasks.

Roadmap. Section 2 presents related work. Section 3 describes the COOKBOOK framework, including the procedures for constructing and combining templates. Section 4 presents our empirical results, and section 5 introduces and measures the template alignment statistic, as well as analysis on the role of random tokens and data generating rules. We conclude with a discussion in Section 6.

2 RELATED WORK

We present an abbreviated related work. A full treatment can be found in Appendix B. There is a large body of work which studies instruction-tuning dataset creation, programmatic data generation, and improving LLM generative capabilities. The current state-of-the art of instruction tuning datasets focus on generating data using existing LLMs, reducing manual curation efforts but facing challenges such as privacy and terms of service violations (Achiam et al., 2023; Lian et al., 2023a; Taori et al., 2023; Luo et al., 2023; Li et al., 2023; Daniele & Suphavadeeprasit, 2023). Additionally, synthetic data has proven beneficial for pretraining (via token-level tasks and data with latent structures) and also for enhancing LLMs' understanding and classification abilities through fine-tuning (Wu et al., 2022; Papadimitriou & Jurafsky, 2020; Krishna et al., 2021; Bhatia et al., 2023).

3 COOKBOOK

We set up the problem of constructing templates and combining templates in Section 3.1. We then describe our framework, COOKBOOK: templates for generating programmatic data that can improve model abilities on generative tasks (Section 3.2). In Section 3.3, we show how to combine data from several templates in the multi-task setting.

3.1 SETUP

Constructing templates. We are given a NL generative task T as input. Samples of T have several components. Let $I \in \mathcal{I}$ be the NL instruction for the task, $\boldsymbol{x} \in \mathcal{X}$ denote its

```python
def commonsense_reasoning(overlap_len: int):
  sent, c1, c2 = random.sample(token_ids)
  # inputs: use sent to get c1 and c2
  c1 = c1 + random.sample(sent, k=overlap_len)
  c2 = c2 + random.sample(token_ids, k=overlap_len)
  choices = [c1, c2].shuffle()

  # outputs: use sent and choices to get answer
  ans_idx = argmax([sent ∩ choices[0],sent ∩ choices[1]])
  answer = choices[ans_idx]

  instruction = "Select the correct choice."

  return f"""
  {instruction}
  {sent}
  Choices:
  - {choices[0]}
  - {choices[1]}
  Answer: {answer}"""
```

```python
def matching(noise: float):
  e1 = random.sample(token_ids)
  # inputs: use e1 to get e2
  e2 = (e1.replace(k=1) if rand.random() < 0.5
        else e1.replace(k=noise))

  # outputs: use e1 and e2 to get answer
  answer = 'yes' if e1 ∩ e2 > (1-noise) else 'no'

  instruction = "Determine whether Product A and
  Product B are the same."

  return f"""
  {instruction}
  Product A: {e1}
  Product B: {e2}
  Question: Are Product A and Product B the same?
  Answer: {answer}"""
```

Figure 2: **Sample templates (pseudocode)**. Templates construct the inputs, outputs, and then return a formatted sample. (Left) template for commonsense reasoning which generates two answer choices, where one choice (the answer) has a greater token overlap to the sentence. (Right) template for entity matching, which generates two entities which are labeled a match if their similarity is above a certain threshold.

inputs, and $y \in \mathcal{Y}$ denote the task output. Let $\text{fmt}_T()$ be the task formatter that formats the text sample using $\boldsymbol{x}, y$, adding in the instruction I at the beginning. Using document QA as an example, $\boldsymbol{x} = \{x_1, x_2\}$ where x_1 is the document and x_2 is the question, y is the answer, and $\text{fmt}_T(\boldsymbol{x}, y, I)$ could be equal to `f"Use the document to answer the question. Document: {x1}. Question: {x2}. Answer: {y}"`. Given this information about T, we construct a template G_T, a Python function that generates synthetic samples. The goal is to design the template such that the generated samples can be used to fine-tune a model that does well on T. More precisely, by fine-tuning a base model $f : \mathcal{X} \to \mathcal{Y}$ on n samples generated from G_T to yield $f_{G_T,n}$, our goal is for $f_{G_T,n}$ to outperform f on T.

Combining templates. We have as input l templates $G_{\boldsymbol{T}} = \{G_{T_1}, \ldots, G_{T_l}\}$ generated using COOKBOOK for tasks $T_1, \ldots, T_l$. We use these templates to generate n samples of programmatic data with mixture ratio $p = \{p_1, \ldots, p_l\}$. Define $f_{G_{\boldsymbol{T}},np}$ as the base model f fine-tuned on np_i samples generated by G_{T_i} for all $i \in [l]$. Now, suppose there are m downstream evaluation tasks $\boldsymbol{T}^{\text{eval}} = \{T_1^{\text{eval}}, \ldots, T_m^{\text{eval}}\}$ (note that they may be different from the l tasks that the templates were constructed for), and let $\text{acc}(f_{G_{\boldsymbol{T}},np}, T_j^{\text{eval}})$ be the accuracy of $f_{G_{\boldsymbol{T}},np}$ on T_j^{eval}. Our goal is to determine the p that maximizes $f_{G_{\boldsymbol{T}},np}$'s average accuracy, $\underset{p \in \triangle^l}{\text{maximize}} \frac{1}{m} \sum_{j=1}^{m} \text{acc}(f_{G_{\boldsymbol{T}},np}, T_j^{\text{eval}})$.

3.2 COOKBOOK TEMPLATES

We first describe the process by which a template generates a sample. Then, we present several examples of COOKBOOK templates (7 total), explaining how their task-specific inputs and outputs are defined using common operators.

Data generation process The data generation process involves (1) constructing the inputs, $\hat{\boldsymbol{x}}$, and (2) constructing the outputs $\hat{y}$ based on the inputs. The inputs are categorized into a *parent input* and *child inputs*; the parent input is a randomly sampled sequence of tokens from the vocabulary, and the child inputs are constructed from the parent input. The output is constructed from the parent and child inputs using a function that approximates the task rule. The sample is then formatted as $\text{fmt}_T(\hat{\boldsymbol{x}}, \hat{y}, I)$.

3.2.1 EXAMPLE TEMPLATES

We present example templates from three generative *task families*: selection, search, and comparison, described below.

- **Selection**: Selection tasks involve choosing one of the given inputs (answer choices) as an output. Examples include entity disambiguation, commonsense reasoning QA and multiple choice QA.

- **Search**: Search tasks require extracting part of the input. Examples include document QA and retrieval.
- **Comparison**: Comparison tasks involve outputting "yes" or "no" based on if some relation among the inputs is present. An example is entity matching.

For each example template, we describe its inputs (parent, child) and outputs, as well as pseudocode for the template's data generation procedure. Across the tasks, we identify five operators for generating inputs: `random.sample()`, `random.shuffle()`, `get_span()` (extract a random span from a sequence of tokens), `replace(k)` (replace tokens in a sequence with probability k), and list concatenation. Additional task templates can be found in Appendix A.

Document QA (Search) The task of document QA involves answering a question over a provided textual context. We describe key components of the DOCUMENT-QA template below and show the template in Figure 1.

- **Inputs**: "Document" is the parent input, and "Question" is the child input. "Document" is generated as a sequence of randomly sampled tokens from the token vocabulary. "Question" is generated as a subspan of "Document".
- **Outputs**: "Answer" is the output which is generated by locating the "Question" in the "Document" and returning the tokens within k indices of the location.
- **Data generator**: See "doc_QA_template" in Figure 1.

Commonsense Reasoning (Selection) The task of commonsense reasoning QA is to select the answer choice which best completes the sentence. For example, given a sentence of "eating the most sweets" and two choices of (1) "eat more vegetables" and (2) "eat more candy", the correct answer choice is "eat more candy". We describe the key components of COMMONSENSE-SELECT below and show the template in Figure 2.

- **Inputs**: "Sent" is the parent input, and "c1", "c2" ("choices") are the child inputs. "Sent" is generated as a sequence of randomly sampled tokens from the token vocabulary. Both "c1" and "c2" are generated as sequences of randomly sampled tokens from the token vocabulary. For "c2" we append additional tokens which is a sequence of tokens sampled from "Sent". "Choices" is the list containing "c1" and "c2" which is randomly shuffled.
- **Outputs**: "Answer" is the index of "choices" which has the greatest overlap with "Sent".
- **Data generator**: See "commonsense_reasoning" in left of Figure 2.

Entity Matching (Comparison) The task of entity matching is to determine whether two entities are equivalent. We describe the key components of MATCHING below and show the template in Figure 2.

- **Inputs**: The first entity "e1" is the parent input, and the second entity "e2" is the child input. "e1" is generated as a sequence of randomly sampled tokens from the token vocabulary. With 50% probability, "e2" is set to "e1" with a small amount of tokens replaced (as determined by the "noise" threshold); otherwise, "e2" is generated as a sequence of randomly sampled tokens from the token vocabulary of the same length as "e1".
- **Outputs**: "Answer" is determined by the amount of overlap between "e1" and "e2". If the amount of overlap is above a threshold, the outut is "yes". If not, it is "no".
- **Data generator**: See "matching" in right of Figure 2.

Note that across all tasks, outputs are constructed from inputs based on token overlap among the inputs. However, we do not claim that the token overlap and the five operators above are necessary components of a template; in section A.7 we present a poetry generation task and template, which does not invoke most of these components.

3.3 COMBINING TEMPLATES

While we previously discussed how a template is constructed to help a model learn a single task, in practice we often want models to improve on many tasks at once. We study how to set the proportion by which we combine the programmatic data across several templates into a training dataset. First,

Model	arc_c	arc_e	boolq	hellaswag	lambada	openbookqa	piqa	winogrande	average
LLAMA-2-7B	46.25	74.58	77.74	75.99	73.92	44.20	79.11	69.14	67.61
LLAMA-2-7B-NH	49.74	76.09	80.00	77.72	72.99	**46.40**	79.76	70.01	69.09
MISTRAL-7B	54.10	79.50	83.49	81.12	75.59	44.40	82.05	73.88	71.76
MISTRAL-7B-ORCA	56.14	79.59	86.57	81.73	72.37	45.60	**83.03**	73.24	72.28
MISTRAL-7B-OH	**59.98**	81.65	**86.73**	**81.77**	73.90	44.20	82.70	73.56	73.06
COOKBOOK-LLAMA	48.04	76.77	79.20	76.04	77.10	43.40	78.56	69.30	68.55
COOKBOOK-MIST	57.76	**83.21**	85.23	80.99	**78.23**	44.00	82.32	**74.27**	**73.25**

Table 1: **Performance on GPT4ALL benchmark.** We denote our COOKBOOK-tuned MISTRAL-7B model that uses our data proportions algorithm as COOKBOOK-MIST. Averaged across tasks, COOKBOOK-MIST is the best. For baseline datasets, "NH" denotes NousHermes, "OH" is Open-Hermes, and "ORCA" is OpenOrca.

using a simple linearity assumption, we derive the optimal data proportions $p^\star$ that maximizes average downstream accuracy of the COOKBOOK-tuned model. This approach requires evaluating models on labeled data, so we present an extension where we can approximate $p^\star$ using unlabeled data via a latent variable model inspired by weak supervision Ratner et al. (2019).

Optimal template-generated data proportions. Recall that our objective is to determine the p that maximizes $f_{\boldsymbol{T},np}$'s average downstream accuracy, $\underset{p\in\triangle^l}{\text{maximize}} \frac{1}{m}\sum_{j=1}^{m} \text{acc}(f_{G_{\boldsymbol{T}},np}, T_j^{\text{eval}})$. To solve this problem, we impose a simple linear assumption: that $\text{acc}(f_{G_{\boldsymbol{T}},np}, T_j^{\text{eval}}) = \sum_{i=1}^{m} p_i \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}})$ for all $i \in [l]$. That is, the accuracy of a model trained on a weighted mixture is equal to the weighted average of models trained on each template, which we empirically assess in Appendix C.2. We also add an entropy term $H(p) = -\sum_{i=1}^{l} p_i \log p_i$ with a weight $\eta \geq 0$ to control how close to uniform p should be. Our optimization problem is now

$$\underset{p\in\triangle^l}{\text{maximize}} \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{l} p_i \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}}) + \eta H(p). \tag{1}$$

Solving this optimization problem yields the following solution.

Proposition 1. *Define* $A \in \mathbb{R}^{l\times m}$ *where* $A_{ij} = \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}})$. *Let* $\sigma_i = \exp(\frac{1}{m\eta}\sum_{j=1}^{m} A_{ij})$ *for all* $i \in [l]$. *Then, the* $p^\star$ *that maximizes* (1) *is* $p_i^\star = \frac{\sigma_i}{\sum_{k=1}^{l}\sigma_k}$ *for all* $i \in [l]$.

See Appendix C.3 for the proof. The computation of $p^\star$ is straightforward. We train one model per template, $f_{G_{T_i},n}$ for each $i \in [l]$. We compute the average accuracy across the m downstream tasks per model, and then compute the softmax over these accuracies to get the proportions $p^\star$ for how many samples are needed from each template. See Algorithm 1 in Appendix C.1.

Extension to evaluation data without ground-truth outputs. Note that computing $\text{acc}(f_{GT_i,n}, T_j^{\text{eval}})$ requires evaluating on a dataset with ground-truth outputs. However, in practice datasets are not often annotated with outputs, which requires us to estimate the accuracy of the model. Since we have multiple individual models, we can frame them as noisy "voters" and cast accuracy estimation as a weak supervision problem, a line of work that estimates labels given an unlabeled dataset and several heuristic voters using latent variable models.We describe this extension of our approach, which uses the MeTaL weak supervision algorithm (Ratner et al., 2019) in Appendix C.1.

4 EXPERIMENTAL EVALUATIONS

We evaluate the empirical performance of COOKBOOK in a general multi-task evaluation setting and a single-task evaluation setting. Then, we show that COOKBOOK can be extended to creative tasks (i.e., poetry generation) going beyond select, search, and compare tasks.

4.1 MULTI-TASK EVALUATION

We evaluate if the COOKBOOK framework can improve multiple generative capabilities at once. Here, we compare COOKBOOK to SoTA instruction tuning datasets (e.g., OpenOrca, OpenHermes).

Our method. We fine-tune two pre-trained base models (LLAMA-2-7B (Touvron et al., 2023) and MISTRAL-7B (Jiang et al., 2023)). The set of templates we consider is: MATCHING, ENTITY-DISAMBIGUATION, MULTI-CHOICE-QA, and COMMONSENSE-SELECT (refer to Appendix A for their constructions). We fine-tune each of the base models on the aforementioned templates using the data proportions obtained using the approach in Section 3.3 (see Algorithm 1, 2).

Evaluation datasets. We run our evals on the standard GPT4ALL (NomicAI) benchmark considered by many models (e.g., Mamba (Gu & Dao, 2023), Cerebras-GPT (Dey et al., 2023)). We follow the standard evaluations for the benchmark reporting average accuracy for all tasks. We use the standard EleutherAI LM Evaluation Harness library (LM eval) (Gao et al., 2023), for zero-shot evaluations over the GPT4ALL benchmark. The LM eval harness provides a fixed set of prompt formats for all tasks, generating accuracy and standard deviation metrics.

Baselines. As baselines we consider the base LLAMA-2-7B and MISTRAL-7B models. Additionally, we compare against the current SoTA open-source instruction-tuned versions of these models (OpenHermes-Mistral-7B (Teknium, 2023), Mistral-7B-OpenOrca (Lian et al., 2023b)) and the closed source versions (Nous-Llama-2-7b (Research, 2023b), Nous-Capybara-7B (Research, 2023a)).

Results. Results for our multi-task evaluations can be found in Table 1. Averaging across tasks we find that (1) our mixtures improve performance across model variants—Llama and Mistral (see Table 1), (2) the template combining approach with no downstream ground-truth outputs described in Section 3.3 is the best fine-tuned model on the benchmark suite (see COOKBOOK-MISTRAL and Table 7), and (3) COOKBOOK-MISTRAL is the best performing model on 3/8 tasks.

4.2 SINGLE-TASK EVALUATION

This section evaluates single-task performance improvements from task-specific templates. We consider two 1B parameter models: GPT-NEO-1.3B (Gao et al., 2020) and CEREBRAS-GPT-1.3B (Dey et al., 2023).

DS	CEREBRAS-BASE	CEREBRAS-FEW	CEREBRAS-FT	COOKBOOK-CEREB
TYDIQA	9.90 ± 0.48	8.20 ± 1.85	**42.30 ± 0.90**	37.50 ± 1.19
SQUAD	32.10 ± 0.78	34.00 ± 3.40	**64.80 ± 2.20**	51.30 ± 1.00
PIQA	0.00 ± 0.00	47.50 ± 0.61	45.60 ± 0.79	**52.80 ± 0.13**
MS_MARCO	6.60 ± 0.49	14.40 ± 1.43	**19.10 ± 0.62**	18.70 ± 1.09
WINOGRANDE	0.17 ± 0.13	30.60 ± 20.00	50.90 ± 1.13	**60.30 ± 0.79**
BEER	18.40 ± 0.00	26.60 ± 0.00	26.60 ± 0.00	**74.10 ± 0.00**
ITUNES-AMAZON	40.00 ± 0.00	39.90 ± 0.28	44.00 ± 0.00	**55.40 ± 0.00**

Table 2: **Single-task evaluations**. Values reported are accuracy for all datasets with the exception of BEER and ITUNES-AMAZON, where we report F1-score. COOKBOOK-tuned models are competitive with finetuned models.

Our method. For each evaluation task, we generate data from the associated COOKBOOK template (see Table 6 for the mapping between task and template) and fine-tune the base pre-trained model using data generated from the template. In our data generating process, a new batch is sampled at each step. Across tasks, we fine-tune models over an average of 4.2K datapoints.

Evaluation datasets. For our evaluations, we consider 7 tasks across the selection (PIQA, WINOGRANDE), search (TYDIQA, SQUAD, MS_MARCO) and comparison (BEER, ITUNES-AMAZON) categories. For all tasks excluding BEER and ITUNES-AMAZON, for which we report F1-score, we report accuracy. Table 5 in Appendix D.9 contains the statistics for the datasets.

Baselines. We compare our COOKBOOK-tuned models to three baselines: zero-shot, few-shot (where k=3) in-context learning (ICL), and fine-tuning using task specific examples (128 samples). Details of the hyperparameters used to train the baselines can be found in Appendix D.9.

Results. Table 2 shows the results for the CEREBRAS-GPT-1.3B model. We defer the results for the GPT-NEO-1.3B model to Table 8 in Appendix D.9. Our results show that (1) COOKBOOK-CEREB out-performs the zero-shot, ICL and fine-tuning baselines on 4/7 tasks, (2) performance of COOKBOOK models can be 60.1 points better than base zero-shot performance, suggesting that while the models were previously unable to do the task, they are now performant, (3) similar trends can be seen across model families (see Appendix D.9).

4.3 POETRY GENERATION

In this section, we show that the COOKBOOK framework can be extended beyond the three task families to more creative tasks such as poetry generation.

Task. We create a poetry generation NL task wherein the model is prompted to write a poem with a prespecified rhyme structure, say ABAB. We construct 30 topics covering a wide range of subjects (e.g., shoe or cup) that the models are then prompted to generate a poem about. We measure how well the model is able to follow the specified rhyme pattern when generating the poem.

Our method. We fine-tune the MISTRAL-7B model using a template designed for this poetry task; see Appendix A for the exact template and examples of datapoints. We fine-tune the model for 500 steps with a batch size of 64 where each batch uses fresh samples from the templates. Our template uses a dictionary of rhyme words, but does not use any examples of complete poems.

Results. When prompted to generate a 4 line poem with an ABAB rhyme scheme (given 2 in-context examples), base MISTRAL-7B has 20% accuracy, COOKBOOK-POEM has 60% accuracy and GPT-4 has 50% accuracy. Sample poems can be found in Appendix D.2.

5 ANALYSIS OF COOKBOOK

First, to better understand how COOKBOOK-tuned models improve task performance by better adhering to template rules, we propose the template alignment statistic and apply it to several tasks. Then, we analyze the key components of COOKBOOK—namely the random tokens and data generating functions — to better understand the effectiveness of the method.

5.1 TEMPLATE ALIGNMENT FRAMEWORK

We introduce the template alignment scorer, which measures how similar a NL sample and a template are. We use this to define the template alignment statistic, which measures how better adherence to the template's rule is responsible for improved performance on the task.

Template Alignment Scorer Given a task T and its corresponding template G_T as constructed in Section 3, we propose an alignment scorer $s_{G_T} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$. The template-specific scorer captures how well the input sample's relationship between $\boldsymbol{x}$ and y follows that of G_T's data generating functions. We provide three examples below:

- **Document QA**: recall that the template selects a random span of a document as the question and defines the answer as the span surrounding the question. s_{G_T} scores a sample by finding where the sample's answer is located in the document and computing the fraction of words in the question that are near the answer. Note that template-generated samples have a score of 1.
- **Commonsense Reasoning**: recall that the template generates two answer choices, with only one choice containing a substring of the sentence. s_{G_T} scores a sample by finding the words that are unique to the two choices, measuring the minimum normalized embedding distance between each choice's unique words and the words in the goal, and computing the absolute value of the difference between the two choices' minimum distances to the input sentence. This quantity is large if one answer choice has much higher word similarity to the input sentence than the other. Note that when we use the hamming distance, template-generated samples have a score of 1.
- **Entity Matching**: recall that the template first generates one entity randomly, and then generates the equivalent second entity to have overlap with the first one. s_{G_T} scores a sample by simply computing the word overlap between the two entities, which corresponds to how the template generates equivalent entity pairs. Note that template-generated samples have a score equal to $1 -$ noise_parameter when the entities are equivalent.

Template Alignment Statistic Next, we use the alignment scorer to relate model performance to template rule adherence. Recall the base model $f : \mathcal{X} \to \mathcal{Y}$ and the COOKBOOK-tuned model $f_{G_T,n}$. Given task samples $\mathcal{D}_T$, let $\mathcal{D}_T^+(G_T) = \{(\boldsymbol{x}, y) \in \mathcal{D}_T : f(\boldsymbol{x}) \neq y, f_{G_T,n}(\boldsymbol{x}) = y\}$ be the set of samples that f gets incorrect and $f_{G_T,n}$ gets correct, and let $\mathcal{D}_T^-(G_T) = \{(\boldsymbol{x}, y) \in \mathcal{D}_T : f(\boldsymbol{x}) \neq y, f_{G_T,n}(\boldsymbol{x}) \neq y\}$ be the set of samples that both f and $f_{G_T,n}$ get incorrect. We now define the template alignment statistic.

Definition 1. *Let* $S_T^+(G_T) = \{s_{G_T}(\boldsymbol{x}, y) \ \forall (\boldsymbol{x}, y) \in \mathcal{D}_T^+(G_T)\}$ *and* $S_T^-(G_T) = \{s_{G_T}(\boldsymbol{x}, y) \ \forall (\boldsymbol{x}, y) \in \mathcal{D}_T^-(G_T)\}$ *be the alignment scores over* $\mathcal{D}_T^+(G_T)$ *and* $\mathcal{D}_T^-(G_T)$*, respectively. Define* $F_{T,G_T}^+ : [0, 1] \to [0, 1]$ *as the empirical CDF over* $S_T^+(G_T)$ *and similarly define* F_{T,G_T}^-.

Then, the template alignment statistic for task T, template G_T is

$$\mathcal{A}(T, G_T) = \sup_{s \in [0,1]} |F^+_{T,G_T}(s) - F^-_{T,G_T}(s)|. \tag{2}$$

Note that $\mathcal{A}(T, G_T)$ is the total variation distance between the empirical distributions of the template alignment scores for $\mathcal{D}^+_T(G_T)$ and $\mathcal{D}^-_T(G_T)$, as well as the Kolmogorov-Smirnov test statistic for the two-sample hypothesis test on whether the scores for these two subsets of $\mathcal{D}_T$ come from the same distribution. This statistic is thus bounded between 0 and 1. A large value for this statistic suggests that there is significant difference between improved and non-improved samples in terms of how they adhere to template rules.

Template G_T / Task T	**TEMPLATE-MISTRAL $\mathcal{A}(T, G_T)$**
PIQA / COMMONSENSE-SELECT	$(0.867, 7.1 \times 10^{-46})$
DBLP-ACM / MATCHING	(1.0, 0.0025))
TYDIQA / DOCUMENT-QA	(0.615, 0.013)

Table 3: **Template alignment statistics.** Template alignment statistics for COOKBOOK-tuned models. Values are the (template alignment statistic, p-value). COOKBOOK-tuned models have learned the rule captured by the synthetic.

Results. We compute the template alignment statistic for the MISTRAL-7B-template models on the tasks — TYDIQA, PIQA and DBLP-ACM— using the DOCUMENT-QA, COMMONSENSE-SELECT and MATCHING templates respectively. Note that for PIQA, we use sentence-bert embeddings to compute embedding distance between the words in the answer choices and the words in the goal. We find that the alignment statistics (Table 3) are statistically significant (< 0.05), demonstrating that rules taught by the templates are in-fact being applied to the downstream NL tasks.

5.2 BREAKING DOWN RANDOM TOKENS AND RULES

When do random tokens work? The phenomenon that training a model on random tokens improves NL may appear surprising. To better understand it, we hypothesize that this phenomenon only occurs when the model already has a sufficient understanding of NL. To evaluate this, we use PYTHIA-1B checkpoints (log scale from 0 to 144000) as the base models and compare these to models trained from each checkpoint on template-generated data. We do this procedure on PIQA, ITUNES-AMAZON, SQUAD, and WINOGRANDE, finding that there exists a point in model training at which COOKBOOK starts to help performance significantly, before which it has little effect (see Figure 4 in Appendix E). This suggests that the generalization from random tokens to NL is a model-dependent phenomenon.

Do skills taught over random tokens result in less overfitting? We compare the *overfitting* of skills taught over random tokens versus skills taught over NL tokens. We do so by evaluating the performance of a MATCHING COOKBOOK-tuned model across three tasks and the performance of a model trained on the NL matching task itself (NL-tuned) (see Table 10). We observe that COOKBOOK-tuning actually improves base model performance across tasks beyond matching (up to 5 points), whereas NL-tuning worsens base performance (by up to 7 points) for all tasks outside of matching. This suggests that skills taught in the random token space lead to less overfitting.

Do we need data generating functions: Are random tokens all we need? We empirically inspect the importance of the template rules vs. the task format (fmt_T) in improving downstream performance by replacing our rule-based generated input and outputs in $\text{fmt}_T()$ with random tokens. Our results show that finetuning on data without rules leads to worsened performance — an average performance drop of 18.3 accuracy points (see Table 11 in Appendix E).

6 DISCUSSION

In this work, we tackle the challenge of data curation for improving generative abilities. Via COOKBOOK, we show that it is possible to programmatically generate data that teaches task-specific rules without raising any legal or privacy concerns. Moreover, we show how such programmatic data can be combined to teach multiple skills at once. Finally, we propose a method for measuring whether a model has indeed learned a task rule, and whether learning that rule improves downstream task performance. In future work, we seek to understand how to automate the template creation process.

REFERENCES

Winogrande: An adversarial winograd schema challenge at scale. 2019.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv e-prints*, pp. arXiv–2312, 2023.

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

Kush Bhatia, Avanika Narayan, Christopher De Sa, and Christopher Ré. Tart: A plug-and-play transformer module for task-agnostic reasoning. *arXiv preprint arXiv:2306.07536*, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, pp. 2, 2023.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

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 *NAACL*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Luigi Daniele and Suphavadeeprasit. Amplify-instruct: Synthetically generated diverse multi-turn conversations for effecient llm training. *arXiv preprint arXiv:(coming soon)*, 2023. URL https://huggingface.co/datasets/LDJnr/Capybara.

Nolan Dey, Gurpreet Gosal, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, Joel Hestness, et al. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023.

Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

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. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*, 1992. URL https://aclanthology.org/C92-2082.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Kundan Krishna, Jeffrey P Bigham, and Zachary C Lipton. Does pretraining for summarization require knowledge transfer? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3178–3189, 2021.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large scale language model society, 2023.

Wing Lian, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". Openorca: An open dataset of gpt augmented flan reasoning traces. https://https://huggingface.co/Open-Orca/OpenOrca, 2023a.

Wing Lian, Bleys Goodson, Guan Wang, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". Mistralorca: Mistral-7b model instruct-tuned on filtered openorcav1 gpt-4 dataset. https://huggingface.co/Open-Orca/Mistral-7B-OpenOrca, 2023b.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 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. *arXiv preprint arXiv:2306.08568*, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3470–3487, 2022.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2022.

NomicAI. Gpt4all performance benchmark. https://gpt4all.io/index.html.

Isabel Papadimitriou and Dan Jurafsky. Learning Music Helps You Read: Using transfer to study linguistic structure in language models. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6829–6839, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.554. URL https://aclanthology.org/2020.emnlp-main.554.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Alexander Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly, 2016.

Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. Training complex models with multi-task weak supervision. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014763. URL https://doi.org/10.1609/aaai.v33i01.33014763.

Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal*, 29(2-3):709–730, 2020.

Shauli Ravfogel, Yoav Goldberg, and Tal Linzen. Studying the inductive biases of RNNs with synthetic variations of natural languages. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3532–3542, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1356. URL https://aclanthology.org/N19-1356.

Nous Research. Nous-capybara-7b-v1.9, 2023a. URL https://huggingface.co/NousResearch/Nous-Capybara-7B-V1.9. Hugging Face Model.

Nous Research. Onousresearch-llama-2-7b-hf, 2023b. URL https://huggingface.co/NousResearch/Llama-2-7b-hf. Hugging Face Model.

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

"Teknium". Openhermes. https://huggingface.co/datasets/teknium/openhermes, 2023.

Teknium. Openhermes-2.5-mistral-7b, 2023. URL https://huggingface.co/teknium/OpenHermes-2.5-Mistral-7B. Hugging Face Model.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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, 2022a.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109, 2022b.

Yuhuai Wu, Felix Li, and Percy S Liang. Insights into pre-training via simpler synthetic tasks. *Advances in Neural Information Processing Systems*, 35:21844–21857, 2022.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

A TEMPLATES

A.1 MATCHING

Data generating template:

```
def matching(noise: int):
    import random

    e1 = random.sample(token_ids, k=1)

    # inputs: use e1 to get e2
    e2 = e1.replace(k=1) if random.random() < 0.5 else e1.replace(k=noise
    ↪ )

    # outputs: use e1 and e2 to get output
    answer = 'yes' if len(set(e1) & set(e2)) > (1 - noise) * len(e1) else
    ↪  'no'
    instruction = "Determine whether product A and product B are the same
    ↪ .\n"

    return f"""
        {instruction}
        Product A: {e1}
        Product B: {e2}
        Question: Are Product A and Product B the same?
        Answer: {answer}"""
```

Sample data point:

```
Determine whether product A and product B are the same
Product A:  occur Competing TObuiltembean Hollywood met
Product B:  occur Competing TObuiltembean Hollywood met
Question: Are Product A and Product B the same?
Answer: yes
```

A.2 MULTI-CHOICE QA

Data generating template:

```
def multi_choice_qa (overlap_len: int):
  question = random.sample(token_ids)
    (c1, c2, c3, c4, c5) = random.sample(token_ids, k=5)

  # inputs: use question to get correct choice c5
    c5 = question.sample(k=overlap_len) + c5[:-overlap_len]

  # outputs: use  question, [c1, . . ., c5]) to get the answer
    choices = [c1, c2, c3, c4, c5].shuffle()
  ans_idx = argmax([question \cap choices[0],..., question \cap choices
    ↪ [4]])
  answer = choices[ans_idx]

  instruction = "Answer the question.\n"
  return f"""
  {instruction}
  Question: {question}
  Choices:
  - {choices[0]}
  - {choices[1]}
  - {choices[2]}
  - {choices[3]}
```

```
  - {choices[4]}
  Answer: {answer}"""
```

Sample data point:

```
Answer the question.
Why lake Shares wildly Gandhi rers ademic AES 1995 ports?
Choices:
- poke installment
- ORS> unmanned Slave ellar Mart English
- visions AES wildly lakeexper Gamer Gate ports ademicE
- Haibeh Dom
- aiming 462 adultery Greenberg collar
Answer:  visions AES wildly lakeexper Gamer Gate ports ademicE
```

A.3 DOCUMENT QA

Data generating template:

```
def qa_template(min_slen: int, max_slen: int):
  document = random.sample(token_ids) #input source

  # inputs: use document to get question
  span_length = random.choice(min_slen, max_slen)
  question = document.get_span(k=span_length) # get indexes


  # outputs: use document and question to get the answer
  answer = document[loc(document.intersect(question)) - 3 : loc(document.
    ↪ intersect(question)) + 3]

    instruction = "Use the document to answer the question.\n"

    return f"""{instruction}
  Document: {document}

  Question: {question}
Answer: {answer}""
```

Sample data point:

```
Use the document to answer the following question.
Document:  Destiny Ricardo Bundle swarm trips spilling crews
trips Python disliked absorption phon Fallen Mour Wales
parameter

Question:  swarm trips spilling
Answer: estiny Ricardo Bundle swarm trips spilling crews trips
Python
```

A.4 ENTITY DISAMBIGUATION

Data generating template:

```
def entity_disambiguation(e_span_len: int):
  sentence_1, sentence_2 = random.sample(token_ids)

  # inputs: use sentence_1 to set sentence_2, e1 and e2
  s1, s2 = sentence_1.get_span(k=e_span_len)
```

```
  e1, e2 = s1[0], s2[0]


  if random.rand() < 0.5
    sentence_2 +=['<blank>'] + s1[1:])
  else:
    sentence_2 +=['<blank>'] + s2[1:])


  # outputs: use (question, sentence_1, sentence_2) to get the answer
  support = sentence_2.split('<blank>')[-1]
  span_loc = loc(sentence_1.intersect(support) - 1]
  answer = sentence_1[span_loc - 1]

  instruction = "Select the choice which best completes the <BLANK>.\n"
  return f"""{instruction}
  {sentence_1}.{sentence_2}
  Choices:
  - {e1}
  - {e2}
  Answer: {answer}"""
```

Sample data point:

```
Select the phrase which best fills in the <BLANK>.
Sentence:  ranc Islands solid illustrates horsepower furry Pay
early Santiago *)scan output. ographies deals privatization
<BLANK>  Santiago *)scan output.
Choices:
-  Islands
-  early
Answer:  early
```

A.5 COMMONSENSE SELECTION

Data generating template:

```
def commonsense_qa(overlap_len: int):
  question = random.sample(token_ids)
  c1 = random.sample(token_ids)
    c2 = random.sample(token_ids)

  # inputs: use question to set c1 and c2
  c1 = c1 + question.sample(k=overlap_len)
  c2 = c2 + random.sample(token_ids, k=overlap_len)
  choices = [c1, c2].shuffle()

  # outputs: use (question, [c1, . . ., c5]) to get the answer
  ans_idx = argmax([len(question.intersect(choices[0])), len(question.
    ↪ intersect(choices[1]))
  answer = choices[ans_idx]

  instruction = "Answer the question.\n"
  return f"""
  {instruction}
  Question: {question}
  Choices:
  - {choices[0]}
  - {choices[1]}
  Answer: {answer}"""
```

Sample datapoint:

```
Select the choice which best completes the sentence.
midst Gloss Quant Nest engaging Soul Customer
Choices:
- Zack extraordinarily willingly Gene Quant engaging rest
- Zack extraordinarily willingly Gene With No DXwaters shone
Answer: Zack extraordinarily willingly Gene Quant engaging rest
```

A.6 TOKEN RETRIEVAL

Data generating template:

```
def token_retrieval(overlap_len: int):
  docs = []
  for i in range(10):
    docs.append(random.sample(token_id)

  # inputs: use docs to set question
  idx = random.randint(0,10) # randomly sample an index from 0...9
  doc_with_answer = docs[idx]
  question = doc_with_answer.sample(overlap_len)

  #outputs: use (question, docs) to get the answer
  answer_idx = argmax[question.intersect(docs[0]),...,question.intersect(
    ↪ docs[9])
  answer = docs[answer_idx]

  instruction = "Use the documents to answer the question.\n"
  return f"""
  {instruction}
  {docs[0]}
  {docs[1]}
  ...
  {docs[9]}
  Question: {question}
  Answer: {answer}"""
```

Sample datapoint:

```
Use the documents to answer the question.
Document 0: Def culture Luc writers takingo  adjustment
Document 1: saturally bites song house speak
Document 2: BEATE level arbitrator Layer eminent substant
Document 3: spend conesz identification unincorporateddemand
Document 4: incident existent Back Fellow Turk peacea Father
Document 5: citations Cove solel Nick cards removing comprise
Document 6: quilt sun instructed facil enacted council confess
Document 7: cs probably 59 specify Page harris famine pumps
Document 8: equal Originally quick Adjust hearted OCK beat
Document 9: Base Sept 168 '</resh593Mr tang moss mash pain

Question: 593Mr tang moss mash pain Dir
Answer: Base Sept 168 '</resh593Mr tang moss mash pain
```

A.7 POEM SCHEME

Data generating template:

```
def rhyme_scheme(rhyme_dict: dict):
  # inputs
```

```
  topic = random.sample(token_ids, k=1)
  rhyme_scheme_A, rhyme_scheme_B = rhyme_dict.sample(k=2)

  # inputs: use (topic, rhyme_scheme_A, rhyme_scheme_B) to get lines
  lines = []
  for idx in range(5):
    r_word = rhyme_scheme_A.sample() if idx % 2 == 0 else rhyme_scheme_B.
    ↪ sample()

    line = random.sample(token_ids)
    line = line.insert(topic) + r_word
        lines.append(line)

  instruction = "Write a five line poem with an ABABA rhyme scheme about"

    return f"""
      {instruction} {topic}
      {lines[0]}
      {lines[1]}
      ...
      {lines[4]}""
```

Sample datapoint:

```
Write a five line poem with a ABABA rhyme structure about abi
abi abilities Ches 76 Purabul befriend
abi agre Chung Rapt unfit hate redditt
item Struabi pre transcend
Pedro Rescueabi 1080 vines296speed ledet
continents anticip EMP texts sigabi trend
```

B Complete Related Works

There is a large body of work which studies instruction-tuning dataset creation, programmatic data generation, and improving LLM generative capabilities. Here we review some of them which relate most closely with our work.

Instruction Tuning Instruction tuning datasets seek to improve the ability of LLM's to follow instructions (i.e., "Generate a summary of the following news article"). These datasets consist of input and output pairs, where the input is composed of an instruction with some context and the output is the "gold" generation. Early instruction tuning datasets utilized manual data curation to construct these <instruction, input, output> triples for tasks such as summarization (Bai et al., 2022). In an effort to scale the instruction tuning dataset curation process, new datasets emerged which prepend natural language instructions to standard NLP tasks such as classification (Mishra et al., 2022; Longpre et al., 2023; Wang et al., 2022b; Chung et al., 2022) where the input/output pairs previously exists. More recently, instruction tuning datasets have emerged which use existing LLMs (e.g., GPT-4 Achiam et al. (2023)) to generate the data (Lian et al., 2023a; Taori et al., 2023; Luo et al., 2023; Li et al., 2023; Daniele & Suphavadeeprasit, 2023). In this approach, instructions are fed to the LLM (i.e., "Generate a list of 5 animals") and the model generates an output. While this approach sidesteps manual data curation efforts, it suffer from privacy and terms of service violations. COOKBOOK sidesteps these challenges by circumventing the need for both manual or model generated data.

Programmatic data generation: Programmatic data generation has been primarily studied for the automated *labeling of datasets* (Ratner et al., 2020; 2016; Hearst, 1992). These works provide programmatic labels for classification tasks given an unlabeled dataset containing inputs only. Therefore, they are challenging to extend to our setting, which requires us to construct entire samples for generative tasks—both programmatic inputs and outputs, which are often open-ended generations (e.g. DOCUMENT-QA).

Synthetics for Training Prior work has shown the value of synthetic, token level tasks such as set operations as an alternate to natural language data for LLM pretraining Wu et al. (2022). Similarly, other work has demonstrated the efficacy of data with latent structure (i.e., music) in the pretraining phrase Papadimitriou & Jurafsky (2020) as well as for task such as summarizatons (Krishna et al., 2021). Additionally, synthetic data has be shown to improve classification abilities (Bhatia et al., 2023). These works are similar to COOKBOOK in that they utilize non-language based inputs to improve LLM performance, but unlike COOKBOOK, they focus on the pretraining stage.

Synthetics for LLM Understanding Recent works have explore the use of synthetic tasks for understanding the underlying mechanisms of LLMs (Bricken et al., 2023; Nanda et al., 2022; Ravfogel et al., 2019) as well as for understanding and improving architectures (Fu et al., 2022; Arora et al., 2023; Gu & Dao, 2023; Poli et al., 2023). Unlike these prior works which seek to use non-language based tasks to understand LLMs, we use such data to improve LLM capabilities.

C ADDITIONAL DETAILS ON COMBINING TEMPLATES

First, we describe the full algorithm for combining data from templates, with or without labeled evaluation data. Then, we demonstrate that the linear assumption empirically holds. Finally, we provide a proof of Proposition 1.

C.1 TEMPLATE COMBINING ALGORITHM

Algorithm 1 presents our approach for how to compute template data proportions. It computes $\hat{p}$ according to Proposition 1; namely, l COOKBOOK-tuned models—one per template G_{T_i}—are trained, and then evaluated on all m downstream tasks. Then, the average accuracy across tasks for each COOKBOOK-tuned model is computed, and a softmax (with temperature $1/\eta$ depending on the entropy regularization in (1)) is performed over these average accuracies to get $\hat{p}$.

When evaluating each COOKBOOK-tuned model, computing accuracy on each downstream task is straightforward if ground-truth outputs are available for the task. However, if they are unavailable but the outputs are discrete (e.g., answer choices in PIQA or "yes" or "no" in ITUNES-AMAZON), we can use techniques from weak supervision to estimate accuracies (line 6 of Algorithm 1). Weak supervision involves producing programmatically labeled data by modeling several weaker "voters" using a latent variable probabilistic graphical model over their votes. In particular, weak supervision algorithms fit the latent variable model using predictions from the voters, estimating the voter accuracies. Then, they aggregate the votes based on the estimated accuracies to get a programmatic label. We utilize a popular weak supervision algorithm from Ratner et al. (2019) to get an estimated accuracy for each COOKBOOK-tuned model in Algorithm 2. In particular, we compute a votes dataset $\mathcal{D}_\lambda \in \mathcal{Y}^{m \times l}$ by using each COOKBOOK-tuned model to produce predictions on the downstream dataset. This votes dataset is passed in as input to Algorithm 1 of Ratner et al. (2019). This algorithm uses a conditional independence assumption, which implies a particular sparsity pattern in the inverse covariance matrix of $\mathcal{D}_\lambda$, to create a system of equations that is solved with SGD to yield estimated accuracies.

C.2 ASSESSING THE LINEARITY ASSUMPTION EMPIRICALLY

To see if the linearity assumption applies on real data, we consider two templates $G_{\boldsymbol{T}} = \{G_{T_1}, G_{T_2}\}$, and compare the performance of individual COOKBOOK-tuned models, $f_{G_{T_1},n}$ and $f_{G_{T_2},n}$, versus a model fine-tuned on a uniform mixture over the two templates, $f_{G_{\boldsymbol{T}},n[1/2,1/2]}$, which we call the mixture model. We re-use the setting described in Section 4.1 (4 templates, 8 tasks from GPT4ALL, fine-tuning the MISTRAL-7B model). We measure two quantities, described below.

- **Interpolation property:** first, we examine how often the mixed model's accuracy interpolates between the individual COOKBOOK-tuned model accuracies, $\text{acc}(f_{G_{\boldsymbol{T}},n[1/2,1/2]}, T_j^{\text{eval}}) \in [\text{acc}(f_{G_{T_1},n}, T_j^{\text{eval}}), \text{acc}(f_{G_{T_2},n}, T_j^{\text{eval}})]$ on all downstream tasks $T_j^{\text{eval}} \in \boldsymbol{T}^{\text{eval}}$. We compute how many pairs of G_{T_1}, G_{T_2} (out of 6 pairs over the 4 templates) for which this condition holds per evaluation task. Our results are shown in Figure 3 (left), where we find that all downstream tasks have at least half of the template pairs satisfying this interpolation property.

Algorithm 1 Template Combination.

Input: l templates $G_{\boldsymbol{T}}$, base model f, m downstream task datasets $\boldsymbol{\mathcal{D}}^{\text{eval}} = \mathcal{D}_1^{\text{eval}}, \ldots, \mathcal{D}_m^{\text{eval}}$, n number of training samples, η entropy parameter
Fine-tune f on n samples generated from G_{T_i} to get COOKBOOK-tuned model $f_{G_{T_i},n}$ for all templates $G_{T_i} \in G_{\boldsymbol{T}}$.
if $\boldsymbol{\mathcal{D}}^{\text{eval}}$ has ground-truth outputs **then**
    Evaluate each model $f_{G_{T_i},n}$ on $\boldsymbol{\mathcal{D}}^{\text{eval}}$ to get $\hat{A}_{ij} = \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}})$ for all $j \in [m], i \in [l]$.
else
    Set $\hat{A}_{ij} = \text{ESTIMATEACCS}(f_{G_{T_1},n}, \ldots, f_{G_{T_l},n}, \mathcal{D}_j^{\text{eval}})$, a weak supervision-based method for estimating accuracy without ground-truth outputs, for all $j \in [m], i \in [l]$.
end if
Compute $\sigma_i = \exp(\frac{1}{m\eta} \sum_{j=1}^{m} \hat{A}_{ij})$ for all $i \in [l]$.
Calculate sampling proportion vector $\hat{p}$, where $\hat{p}_i = \frac{\sigma_i}{\sum_{k=1}^{l} \sigma_k}$ for all $i \in [l]$.
return $\hat{p}$. f is then fine-tuned on n samples from templates $f_{G_{\boldsymbol{T}}}$ with proportion $\hat{p}$.

Algorithm 2 ESTIMATEACCS.

Input: l COOKBOOK-tuned models ("voters") $\lambda_i := f_{G_{T_i},n} : \mathcal{X} \to \mathcal{Y}$ for $i \in [l]$, dataset without ground-truth outputs, $\mathcal{D} = \{\boldsymbol{x}^j\}_{j=1}^n$.
Get predictions $\{\lambda_1(\boldsymbol{x}^j), \ldots \lambda_l(\boldsymbol{x}^j)\}$ across COOKBOOK-tuned models for each $x_j \in \mathcal{D}, i \in [l]$, forming votes dataset $D_\lambda \in \mathcal{Y}^{n \times l}$.
Estimate accuracy $\hat{a}_i \approx \Pr(\lambda_i(\boldsymbol{x}) = y)$ for all $i \in [l]$ by using Algorithm 1 from Ratner et al. (2019) on noisy votes D_λ.
return $\{\hat{a}_1, \ldots, \hat{a}_l\}$.

- **Mixture model deviation:** second, we measure the absolute difference between the mixture model's accuracy on a task and the average of the individual COOKBOOK-tuned models' accuracies on the task, $|\text{acc}(f_{G_{\boldsymbol{T}},n[1/2,1/2]}, T_j^{\text{eval}}) - \text{avg}(\text{acc}(f_{G_{T_1},n}, T_j^{\text{eval}}), \text{acc}(f_{G_{T_2},n}, T_j^{\text{eval}}))|$. We call this the mixture model deviation, and measure this across template pairs and downstream tasks in Figure 3 (right). We find that most values of this deviation are between 0 and 1, meaning that the mixture model's accuracy is less than 1 accuracy point away from the average of individual accuracies.

Based on these results, we do see that there exist some template pairs for which training a model on them results in significantly different performance. This suggests that modeling higher-order interactions among data samples from different templates could help us improve our estimate of $p^\star$, although doing so may result in an optimization problem that lacks a simple closed-form solution.

C.3 PROOF OF PROPOSITION 1

We restate Proposition 1 here for completeness.

Proposition 1. *Define* $A \in \mathbb{R}^{l \times m}$ *where* $A_{ij} = \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}})$. *Let* $\sigma_i = \exp(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij})$ *for all* $i \in [l]$. *Then, the* $p^\star$ *that maximizes* (1) *is* $p_i^\star = \frac{\sigma_i}{\sum_{k=1}^{l} \sigma_k}$ *for all* $i \in [l]$.

Proof. Recall that the objective we aim to maximize is the expression

$$\frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{l} p_i A_{ij} \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}}) + \eta H(p). \tag{3}$$

where $A_{ij} = \text{acc}(f_{G_{T_i},n}, T_j^{\text{eval}})$ and $p \in \triangle^l$, e.g. $\sum_{i=1}^{l} p_i = 1$. The Lagrangian function is thus

$$\mathcal{L}(p, \lambda) = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{l} p_i A_{ij} + \eta H(p) + \lambda\Big(1 - \sum_{i=1}^{l} p_i\Big) \tag{4}$$

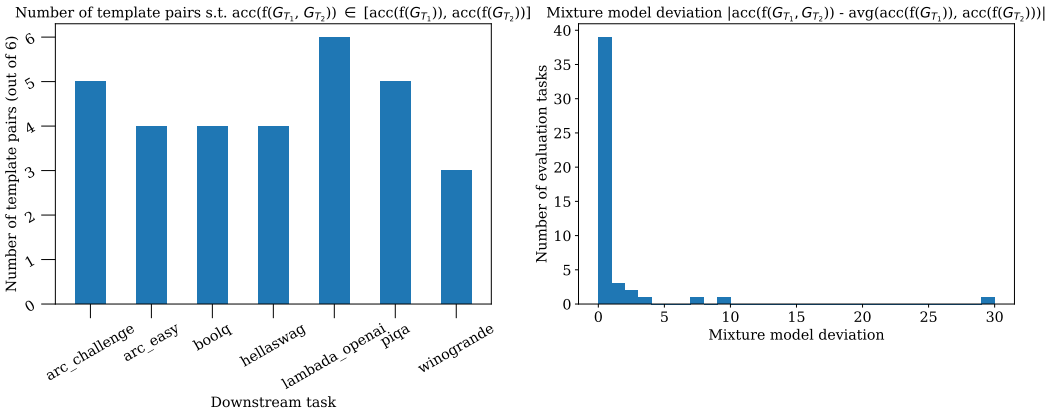

Figure 3: Evaluating if the linearity assumption for data proportions holds empirically. Left: we measure the interpolation property, how often the mixture model has an accuracy in between the individual COOKBOOK-tuned models' accuracies. Right: we measure the mixture model deviation, the absolute difference between the mixture model's accuracy and the average of the individual COOKBOOK-tuned models' accuracies. Measurements are made across 6 pairs of templates (over the 4 templates used in Section 4.1), 8 GPT4ALL evaluation tasks, and the MISTRAL-7B base model.

Taking the partial derivative of the Lagrangian with respect to p_i for some $i \in [l]$ and setting equal to 0, we get

$$\frac{\partial \mathcal{L}(p, \lambda)}{\partial p_i} = \frac{1}{m} \sum_{j=1}^{m} A_{ij} + \eta(-\log p_i - 1) - \lambda = 0 \tag{5}$$

Rearranging, we get

$$\log p_i = \frac{1}{m\eta} \sum_{j=1}^{m} A_{ij} - \frac{\lambda}{\eta} - 1 \tag{6}$$

$$\Rightarrow p_i = \exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij} - \frac{\lambda}{\eta} - 1\right) \tag{7}$$

We now plug this expression for p_i into the equation $\sum_{i=1}^{l} p_i = 1$:

$$\sum_{i=1}^{l} \exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij} - \frac{\lambda}{\eta} - 1\right) = 1 \tag{8}$$

$$\Rightarrow \exp\left(\frac{\lambda}{\eta}\right) = \sum_{i=1}^{l} \exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij} - 1\right) \tag{9}$$

$$\tag{10}$$

Finally, substituting $\exp(\lambda/\eta)$ in the expression for p_i gives us

$$p_i = \frac{\exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij} - 1\right)}{\sum_{i=1}^{l} \exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij} - 1\right)} = \frac{\exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij}\right)}{\sum_{i=1}^{l} \exp\left(\frac{1}{m\eta} \sum_{j=1}^{m} A_{ij}\right)}. \tag{11}$$

□

Model	arc_c	arc_e	boolq	hellaswag	lambada	openbookqa	piqa	winogrande	average
LLAMA-2-7B	46.25 ± 1.46	74.58 ± 0.89	77.74 ± 0.73	75.99 ± 0.43	73.92 ± 0.61	44.20 ± 2.22	79.11 ± 0.95	69.14 ± 1.30	67.61
LLAMA-2-7B-chat	44.20 ± 1.45	69.74 ± 0.94	79.76 ± 0.70	75.50 ± 0.43	71.08 ± 0.63	43.80 ± 2.22	77.20 ± 0.98	66.46 ± 1.33	65.97
LLAMA-2-7B-NH	49.74 ± 1.46	76.09 ± 0.88	80.00 ± 0.70	77.72 ± 0.42	72.99 ± 0.62	**46.40 ± 2.23**	79.76 ± 0.94	70.01 ± 1.29	69.09
MISTRAL-7B	54.10 ± 1.46	79.50 ± 0.83	83.49 ± 0.65	81.12 ± 0.39	75.59 ± 0.60	44.40 ± 2.22	82.05 ± 0.90	73.88 ± 1.23	71.76
MISTRAL-7B-cap	54.01 ± 1.46	78.54 ± 0.84	82.57 ± 0.66	78.74 ± 0.41	72.46 ± 0.62	44.80 ± 2.23	79.60 ± 0.94	71.03 ± 1.27	70.22
MISTRAL-7B-orca	56.14 ± 1.45	79.59 ± 0.83	86.57 ± 0.60	81.73 ± 0.39	72.37 ± 0.62	45.60 ± 2.23	**83.03 ± 0.88**	73.24 ± 1.24	72.28
MISTRAL-7B-OH	**59.98 ± 1.43**	81.65 ± 0.79	**86.73 ± 0.59**	**81.77 ± 0.39**	73.90 ± 0.61	44.20 ± 2.22	82.70 ± 0.88	73.56 ± 1.24	73.06
CB-LLAMA	48.04 ± 1.46	76.77 ± 0.87	79.20 ± 0.71	76.04 ± 0.43	77.10 ± 0.59	43.40 ± 2.22	78.56 ± 0.96	69.30 ± 1.30	68.55
CB-MISTRAL-UNI	58.70 ± 1.44	82.66 ± 0.78	79.97 ± 0.70	81.09 ± 0.39	**78.46 ± 0.57**	43.60 ± 2.22	81.77 ± 0.90	**75.06 ± 1.22**	72.66
CB-MISTRAL +WSL	57.85 ± 1.44	82.37 ± 0.78	86.39 ± 0.60	81.36 ± 0.39	77.94 ± 0.58	44.60 ± 2.23	82.32 ± 0.89	74.19 ± 1.23	73.38
CB-MISTRAL +WS	57.76 ± 1.44	**83.21 ± 0.77**	85.23 ± 0.62	80.99 ± 0.39	78.23 ± 0.57	44.00 ± 2.22	82.32 ± 0.89	74.27 ± 1.23	73.25

Table 4: **Model performance on the GPT4ALL benchmark.** "CB*" denotes our COOKBOOK tuned models where "wsl" is aggregation with labels, "ws" is aggregation w/o labels and "uni" is a uniform mixture, "NH" denotes NousHermes Dataset, "cap" is the Nous-Capybara mdoels, "OH" is Open-Hermes, and "orca" is OpenOrca. Averaged across tasks, CB-MISTRAL +WS is the best performing model.

D EXPERIMENTAL EVALUATIONS: ADDITIONAL DETAILS

D.1 FULL GPT4ALL EVALUATIONS W/STANDARD DEVIATIONS

D.2 POEM-TEMP-MISTRAL SAMPLES

Topic: flower

```
A flower's petals, so delicate and fair
Its scent is sweet, a perfume so divine
Its beauty's rare, a sight to behold anywhere
Its petals fall, a sign of the decline.
```

Topic: bird

```
A bird that soars above the clouds so high
Its wings are strong, its flight is free
It's not afraid to try
And soars above the sea
```

Topic: deer

```
A deer, a female deer, a lovely doe
Its antlers are a symbol of its pride
Amidst the trees, it knows just where to go
To find the food that's on the other side.
```

Topic: galaxy

```
A galaxy of stars, so far away
Their light, a beacon, in the night
A cosmic dance, a celestial display
A wondrous sight, a sight so bright.
```

D.3 HYPERPARAMETER TUNING DETAILS

For our single-task fine-tuning baseline, we run a hyper-parameter search sweeping across learning rate (considering a range $[5e-06, 8e-06, 5e-05, 8e-05]$), batch size ($[32, 64]$) and training epochs ($[3, 4, 5, 10]$).

D.4 GPT4ALL BENCHMARK

GPT4ALL (NomicAI) is a standard evaluation benchmark which covers 7 tasks: ARC-Easy (Clark et al., 2018), ARC-Challenge (Clark et al., 2018), PIQA (Bisk et al., 2020), Winogrande (ai2, 2019), BoolQ (Clark et al., 2019), Lambada OpenAI (Radford et al., 2019), OpenBookQA (Mihaylov et al., 2018) and HellaSwag (Zellers et al., 2019).

D.5 DATASET STATISTICS

We report dataset statistics for the 7 datasets considered in the single-task evaluations. We use the "winogrande-xl" variant of the WINOGRANDE task and an evaluate on v1.1 version of the MS_MARCO dataset. Dataset statistics are listed below in Table 5.

Dataset	Train	Validation	Test
PIQA	16.1K	1.84K	3.08K
SQUAD	87.6K	10.6K	None
TYDIQA	151K	18.7K	None
MS_MARCO	82.3K	10K	9.65K
WINOGRANDE	40.4K	1.27K	1.77K
BEER	80	91	91
ITUNES-AMAZON	156	109	109

Table 5: Summary of dataset sizes for different tasks.

D.6 SINGLE-TASK FINE-TUNING: TEMPLATE TO TASK MAPPING

Below, we map the COOKBOOK templates used to fine-tune models for the corresponding NL task.

Dataset	Template
PIQA	COMMONSENSE-SELECT
SQUAD	DOCUMENT-QA
TYDIQA	COMMONSENSE-SELECT
MS_MARCO	TOKEN-RETRIEVAL
WINOGRANDE	ENTITY-DISAMBIGUATION
BEER	MATCHING
ITUNES-AMAZON	MATCHING

Table 6: Task to template mapping.

D.7 EVALUATION DETAILS

For all single-task evaluations found in Table 2 and Table 8, we evaluate on 1K examples from the tests sets, with the exception of PIQA, SQUAD and TYDIQA where we sample from the validations sets because they either don't have test sets (SQUAD, TYDIQA) or the test sets are unlabeled (PIQA). We run our evaluations across three different test sets, generated using three separate random seeds.

D.8 TEMPLATE COMBINATION EVALUATION DETAILS

Table 7 provides additional experimental results around our approach for combining data from templates. For COOKBOOK-tuned models that use multiple templates, our main method (COOKBOOK plus data proportions obtained using WS on data without ground-truth outputs) is referred to as CB-WS, (previously COOKBOOK-MIST in Table 1). Furthermore, COOKBOOK-WSL shows the results when we use the ground-truth outputs in our evaluation datasets; while this method has slightly higher average accuracy, CB-WS is able to come close despite not having any output information. COOKBOOK-UNI shows the results on a uniform mixture of template-generated data, which performs worse than both previous methods. Table 7 also contains results of the individual models that are COOKBOOK-tuned on MATCHING, ENTITY-DISAMBIGUATION, MULTI-CHOICE-QA,

Model	arc_c	arc_e	boolq	hellaswag	lambada	openbookqa	piqa	winogrande	average
CB-MATCH	57.40 ± 2.21	82.60 ± 1.70	65.20 ± 2.13	72.00 ± 2.00	75.20 ± 1.93	43.80 ± 2.22	81.80 ± 1.73	74.60 ± 1.95	69.08
CB-ED	54.00 ± 2.23	79.60 ± 1.80	81.60 ± 1.73	71.20 ± 2.03	79.60 ± 1.80	44.60 ± 2.23	82.40 ± 1.70	73.20 ± 1.98	70.78
CB-MCQA	55.40 ± 2.22	80.20 ± 1.78	83.60 ± 1.66	71.80 ± 2.01	81.00 ± 1.76	44.40 ± 2.22	83.00 ± 1.68	74.60 ± 1.95	71.38
CB-SELECT	55.60 ± 2.22	79.40 ± 1.81	82.60 ± 1.70	72.00 ± 2.00	80.60 ± 1.77	43.60 ± 2.22	82.60 ± 1.70	74.80 ± 1.94	71.40
CB-UNI	58.70 ± 1.44	82.66 ± 0.78	79.97 ± 0.70	81.09 ± 0.39	78.46 ± 0.57	43.60 ± 2.22	81.77 ± 0.90	75.06 ± 1.22	72.66
CB-WSL	57.85 ± 1.44	82.37 ± 0.78	86.39 ± 0.60	81.36 ± 0.39	77.94 ± 0.58	44.60 ± 2.23	82.32 ± 0.89	74.19 ± 1.23	73.38
CB-WS	57.76 ± 1.44	83.21 ± 0.77	85.23 ± 0.62	80.99 ± 0.39	78.23 ± 0.57	44.00 ± 2.22	82.32 ± 0.89	74.27 ± 1.23	73.25

Table 7: **WS performance analysis** "CB*" denotes our COOKBOOK tuned models on MISTRAL-7B, where -WSL combines templates using downstream datasets with ground-truth outputs, -WS combines templates without ground-truth outputs and -UNI is the uniform mixture. MATCH, ED, MCQA, and SELECT are abbreviations for MATCHING, ENTITY-DISAMBIGUATION, MULTI-CHOICE-QA, COMMONSENSE-SELECT individual COOKBOOK-tuned models.

and COMMONSENSE-SELECT; we find that these models' performance is worse than the models that use data from multiple templates.

For the template combination approach that does not use ground-truth outputs, we use the MeTaL algorithm from Ratner et al. (2019) on each downstream task over 5 random seeds, learning rate $1e-4$, and number of iterations equal to 5000 (except for BOOLQ, PIQA, and WINOGRANDE, which use 2000).

D.9 SINGLE-TASK FINE-TUNING: ADDITIONAL RESULTS

Single-task finetuning results for the GPT-NEO-1.3B model can be found in Table 8.

Dataset	NEO-BASE	NEO-FEW	NEO-FT	COOKBOOK-NEO
TYDIQA	21.8 ± 0.49	14.0 ± 2.4	**46.7 ± 2.08**	41.9 ± 0.40
SQUAD	14.5 ± 0.78	50.4 ± 5.2	**74.7 ± 0.65**	60.5 ± 0.68
PIQA	0.0 ± 0.0	48.5 ± 0.5	52.5 ± 0.64	**52.7 ± 0.43**
MS_MARCO	12.6 ± 1.08	17.7 ± 1.0	**24.5 ± 0.98**	18.6 ± 0.16
WINOGRANDE	3.9 ± 0.25	**64.6 ± 32.4**	58.3 ± 1.30	54.3 ± 0.87
BEER	26.7 ± 0.0	26.7 ± 0.0	36.3 ± 0.0	**66.6 ± 0.0**
ITUNES-AMAZON	39.7 ± 0.0	39.7 ± 0.0	63.1 ± 0.0	**69.6 ± 0.0**

Table 8: Performance comparison of GPT-NEO-1.3B. Accuracy is reported for all tasks with the exception of BEER and ITUNES-AMAZON for which F1-score is reported.

E ANALYSIS: ADDITIONAL EXPERIMENTS

Below, we outline additional experiments conducted to better understand the role of random tokens and rules in COOKBOOK.

E.1 DO SKILLS LEARNED ON RANDOM TOKEN DISTRIBUTIONS TRANSFER TO NATURAL LANGUAGE?

We evaluate transfer ability of rules learned over random tokens, by evaluating the performance of template-tuned MISTRAL-7B models, on template generated data over *natural language* data samples. Concretely, rather than constructing the inputs as random sequence of tokens and outputs as a function of the inputs, the main component we keep from the template is the input-output rule. In doing so, we isolate the random token to NL transfer aspect and eliminate additional noise in our results caused by imperfect rules to true task reasoning transfer. Our findings show that the rules learned over the random token distributions do in-fact transfer (see Table 9), improving over base performance by up to 29.8 points.

	MISTRAL-BASE	COOKBOOK-MISTRAL
NL-TEMPLATE-PIQA	0.044	0.326
NL-TEMPLATE-IA	0.666	0.964
NL-TEMPLATE-TYDIQA	0.016	0.076

Table 9: **Random token template to NL-based template data transfer.** Evaluation of random-token-based COOKBOOK-tuned model on natural-language-based template data. Skills learned over random tokens transfer to NL setting.

E.1.1 WHEN DO RANDOM TOKENS WORK?

Figure 4 shows the effects of pre-training duration on the efficacy of COOKBOOK. Our results indicate that models that have a better understanding of NL (models that are trained longer) have more performance gains from COOKBOOK.

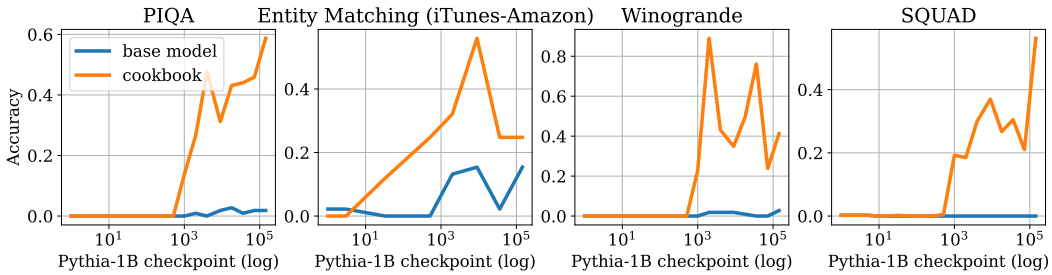

Figure 4: **Effects of pretraining on random token to NL generalization**. Performance gains from COOKBOOK increase with longer pretraining durations, indicating that maturity of NL understanding is correlated with random to NL generalization.

E.1.2 DO SKILLS TAUGHT OVER RANDOM TOKENS RESULT IN LESS OVERFITTING?

Table 10 compares the degree of overfitting experienced a model finetuned on a natural language task itself, and a model finetuned on a COOKBOOK template. Our results indicate that skills taught via the template do not hurt base performance on other tasks, but the skill taught by NL-tuning do.

	MISTRAL-BASE	COOKBOOK-MISTRAL	NL-MISTRAL
PIQA	0.443	0.460	0.412
ITUNES-AMAZON	0.696	0.823	0.875
TYDIQA	0.150	0.200	0.077

Table 10: **Skill overwriting.** Comparison of COOKBOOK and NL-tuned models across tasks. Skill taught by template MATCHING, which corresponds to ITUNES-AMAZON, doesn't hurt base performance on other tasks, but the skill taught by NL-tuning does.

E.1.3 DO WE NEED DATA GENERATING FUNCTIONS: ARE RANDOM TOKENS ALL WE NEED?

Table 11 shows the results of finetuning on a data generated from tempplates with a format but no rule.

	MISTRAL-BASE	COOKBOOK-NORULE-MISTRAL
PIQA	0.464	0.442
ITUNES-AMAZON	0.697	0.195
TYDIQA	0.178	0.151

Table 11: **Templates w/o rule.** Evaluation of MISTRAL-7B tuned on data generated from templates with a fmt_T but no rule.