Better Synthetic Data by Retrieving and Transforming Existing Datasets

Anonymous ACL submission

Abstract

Despite recent advances in large language models, building dependable and deployable NLP models typically requires abundant, high-quality training data. However, task-specific data is not available for many use cases, and manually curating task-specific data is labor-intensive. Recent work has studied prompt-driven synthetic data generation using large language models, but these generated datasets tend to lack complexity and diversity. To address these limitations, we introduce a method, *DataTune*, to make better use of existing, publicly available datasets to improve automatic dataset generation. DataTune performs *dataset transformation*, enabling the repurposing of publicly available datasets into a format that is directly aligned with the specific requirements of target tasks. On a diverse set of language-based tasks from the BIG-Bench benchmark, we find that finetuning language models via DataTune improves over a few-shot prompting baseline by 49% and improves over existing methods that use synthetic or retrieved training data by 34%. We find that dataset transformation significantly increases the diversity and difficulty of generated data on many tasks. We release a Python package and open-source repository to make this method accessible to the community.[1]

1 Introduction

The major bottleneck in modern AI research is data. Despite the paradigm-shifting developments of pre-training and prompting, the recipe for achieving peak performance on any particular task has hardly changed: obtain large amounts of high-quality training data and fine-tune your model. This is particularly valuable for developing models with fewer than 3 billion parameters; in this regime, supervised finetuning can be significantly more effective than in-context learning (Mosbach et al., 2023). However, this recipe can be challenging for specialized or novel tasks where task-specific annotated data is limited.

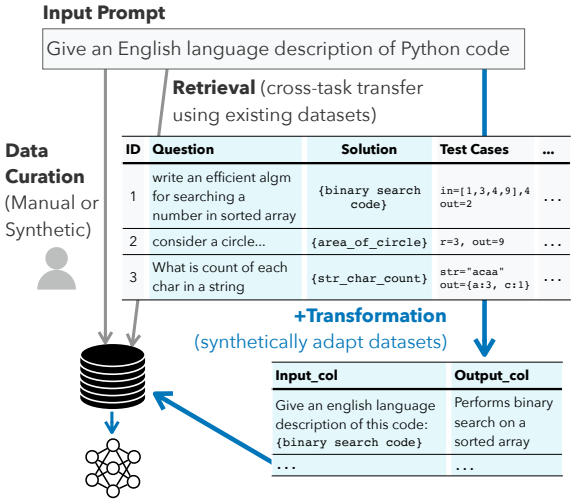

Figure 1: Existing solutions include cross task transfer, which repurposes relevant but task misaligned datasets for text-to-text fine-tuning, and data generation methods that either generate new examples via LLMs, risking low diversity, or employ human annotators, incurring high costs. Our approach synthesizes these strategies by adaptively transforming existing datasets for the target task, preserving original dataset diversity while ensuring the quality of synthetically generated data.

Prior works have proposed strategies for fine-tuning in this low-resource setting. The most intuitive way is *obtain labeled data* through manual curation (Callison-Burch, 2009; Zhang et al., 2022), which assumes access to domain experts and may require considerable financial resources to compensate annotators fairly (Huang et al., 2023). An increasingly popular alternative to large-scale manual annotation is to synthetically produce datasets using existing large models (Wang et al., 2022; Peng et al., 2023; Tang et al., 2023). We will refer to this method as *synthetic data generation*. Unfortunately, directly generating synthetic datasets that simultaneously have high correctness and suf-

[1] URL will be added upon acceptance.

ficient diversity is difficult. Models trained on synthetic datasets are typically significantly worse than those trained on manually curated data when standardizing for dataset size, suggesting that current synthetic dataset generation methods still leave significant room for improvement on dataset quality (Ding et al., 2023).

On the other hand, *cross-task transfer* entirely sidesteps the need for strictly in-domain data by training models either in a multi-task fashion on a wide range of datasets (Sanh et al., 2021) or on task-specific datasets closest to the new target task (Vu et al., 2020). We focus on the latter, which we refer to as *existing data*. Recent methods such as Prompt2Model (Viswanathan et al., 2023b) combine both of the above methods, showing the additive benefits of synthetic data generation and existing datasets for finetuning in few-shot settings. However, they report that even for the machine reading question answering dataset of SQuAD (Rajpurkar et al. (2016), a dataset considered today to be largely "solved", training on *existing data* alone is significantly worse than training on manually or synthetically created data.

To overcome the limitations of these approaches, we introduce DataTune, a system that automatically repurposes public datasets for new tasks. DataTune identifies the most relevant public datasets and uses a large language model to transform them into a format aligned with the target task's needs. For instance, for a task requiring descriptions of Python code in English (as shown in Figure 1), DataTune finds a dataset with programming questions and code solutions (along with other, irrelevant data columns). It transforms the retrieved dataset by using the code solutions as input and generating a synthetic description of the code as output. This approach maintains the original dataset's diversity while matching the task specification, boosting performance by 22 points over baselines. We refer to this synthetic adaptation of publicly available datasets as *Dataset Transformation*.

We evaluate the effectiveness of DataTune on six challenging language-based tasks from the BIG-Bench benchmark (Bench Authors, 2023), which are designed to gauge the system's performance across diverse NLP task categories. When compared to few-shot prompting of the same base model (*Mistral-7B*) without fine-tuning, DataTune-enhanced models improve by an average of 5.2 points on five tasks, demonstrating its value for domain-specific fine-tuning. DataTune also can be used additively with existing synthetic dataset generation approaches, yielding an 8-point improvement over the few-shot prompting baseline. Comparing DataTune with existing methods of synthetic data generation, we find that DataTune often produces more difficult and diverse examples, and on a small sample of data we observe that these benefits do not come at the expense of data correctness.

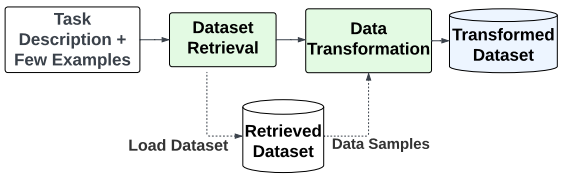

Figure 2: Overview of DataTune. We have two components, one for dataset selection for retrieving a relevant dataset from all existing datasets, and Dataset Transformation for modifying this to match task alignments.

2 Problem Setup

We study the problem of how to automatically *retrieve* and *transform* existing datasets to prepare a fine-tuning dataset for a new task. In our problem setting, a user specifies the task of interest with a textual prompt T (optionally containing a few demonstration examples). We assume access to a large, diverse collection of labeled datasets $\mathcal{D}$, a large language model that be prompted, and a small language model M that we can fine-tune.

The goal here is to automatically generate a synthetic training dataset D' which, after finetuning M on D', will improve M's ability to satisfy the task specification. For each task T in a known set of task descriptions $\mathcal{T}$, we can measure our progress towards this goal by measuring the trained model M_T against labeled data obtained for task T.

3 Methods

DataTune focuses on selecting and transforming a dataset to align it with a specific task. It involves, first, finding relevant datasets that are good candidates for further alignment, through dataset retrieval and reranking; Then, to perform data transformation (i.e., synthetically modify each entries in the selected dataset) for making the dataset better aligned with the task. The LLM used for all components is GPT-4-Turbo, except for the final

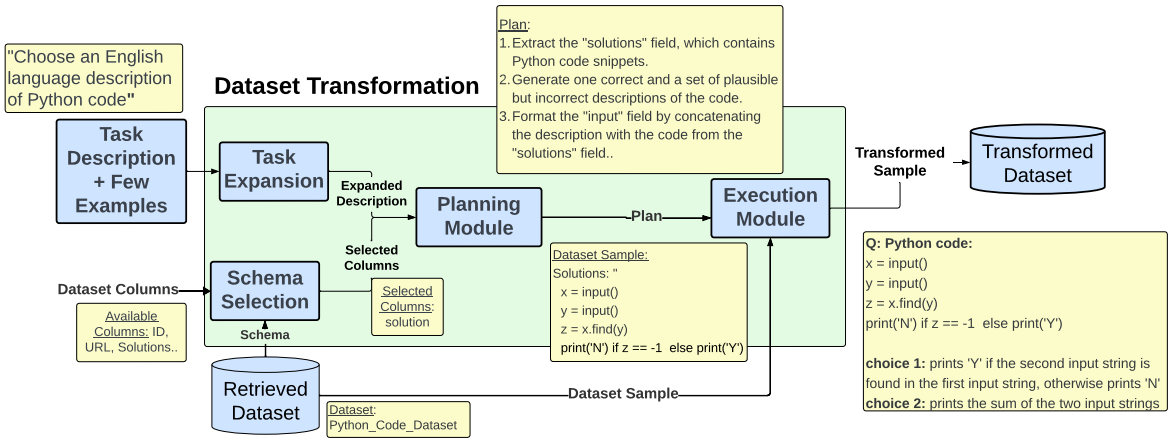

Figure 3: The Data Transformation Component of DataTune, explained with an example (in yellow)

execution step, which uses GPT 3.5-Turbo[2]. We summarize the DataTune architecture in Figure 2.

3.1 Dataset Retrieval

The first phase of DataTune involves identifying relevant datasets for our task. We confine our search to the HuggingFace Hub (Lhoest et al., 2021), which consists of over 75,000 datasets.

To efficiently complete this task, we use a dual-stage retrieval approach. We first *retrieve* a large set of 25 candidate documents using DataFinder (Viswanathan et al., 2023a), a bi-encoder retriever specifically trained for retrieving datasets from natural language queries using textual descriptions of each dataset. We then rerank these 25 candidates to increase the likelihood that our selected dataset can be effectively transformed for the target task. Inspired by Sun et al. (2023), who showed that LLMs can effectively rerank documents, we similarly use LLMs for reranking datasets. We prompt the reranking LLM by providing additional dataset attributes beyond mere descriptions, such as its schema and a small sampling of rows. We include the reranking prompt we used in Appendix A.1. The reranking step may conclude with no suitable dataset being found, acknowledging that not every task has a relevant dataset. To improve the reliability of our selections, we employ self-consistency decoding (Wang et al., 2023), using a voting mechanism to choose the most frequently sampled response from the LLM.

Having selected an appropriate dataset, we now focus on tailoring it to the task requirements. This transformation process can include modifications ranging from aligning the input/output formats to more substantial changes like generating new fields that the present dataset may not currently have.

To accomplish this complex task, we split it into two primary steps. First, the *Planning Module* generate a multi-step plan illustrating the sequence of transformations needed to convert examples from a given dataset into the desired output. Next, the *Execution Module* executes the plan on each datapoint in the dataset.

Empirically, we notice that the Planning Module is more effective when given descriptions of the desired source (dataset) and target (task) that are detailed and specific. As such, we further implement two more scaffolding modules — a *Schema Selector* which removes irrelevant columns in the retrieved datasets (providing a clearer source description) and a *Task Expansion Module* which enriches task descriptions with requirements (giving a better target specification).

Below, we describe the four modules in their execution sequence (examples in Figure 3).

Task Expansion We noticed that the brevity of task descriptions often hampers the creation of an effective plan, and often the subtleties of the task are more readily grasped through the examination of examples. This phenomenon is akin to the process of requirement elicitation in Software Engineering, where a deeper understanding of requirements is achieved through detailed analysis (Lamsweerde, 2009). To address this issue, we implemented an intermediate step wherein the LLM is

[2]Our dataset transformation method requires making an LLM query for each instance in the dataset. Therefore, the decision to use GPT-3.5-turbo was made for budgetary reasons.

Sample Plan

1. Extract the "solutions" field from the dataset as this contains the Python code snippets.
2. For each "solutions" entry, identify the primary operation or functionality of the Python code. This may require parsing the code and understanding its logic.
3. Generate a set of multiple-choice descriptions ("choices") for each code snippet. These should include one correct description of what the code does and several incorrect descriptions. The incorrect descriptions can be plausible but should not accurately describe the code's functionality.
4. Format the "input" field by labeling it as "Python code:" followed by the actual code snippet from the "solutions" field. Below the code, list the generated "choices" with the label "choice:" preceding each option.
5. Determine the correct "choice" that accurately describes the code's behavior. This will be the "output" field.
6. Combine the "input" field and the "output" field to create the final data in the required format for the task examples.
7. If a "solutions" entry does not contain a Python code snippet or is not relevant to the task description, ignore the data sample and return null for that entry.

Figure 4: We show an example plan for the task of providing concise descriptions of Python code. The retrieved dataset contains natural language questions and code solutions. The plan then specifies that the transformation must create the correct description, create incorrect descriptions to create an multiple choice dataset, and format changes required to match the target task examples.

utilized to scrutinize both the task description and provided examples and generate an expanded version of the task description. This enhanced task specification helps in devising a more detailed and actionable plan, encompassing more explicit steps tailored to the task at hand.

Schema Selection The Schema Selection component is designed by instructing LLMs to identify the most pertinent columns within a dataset for a given task. This involves providing the LLM with detailed information, including the task description, the chosen dataset, the names of existing columns, and samples of dataset rows. The LLM is then tasked with identifying which columns are relevant to the specific task at hand. This approach ensures a targeted selection of dataset features that are directly applicable to the task requirements, optimizing the dataset for the intended application. The schema selection prompt is linked in the Appendix A.2.

Planning Module Now we work on developing a comprehensive plan to adapt each data point from the retrieved dataset to the task requirements. This plan is usually a series of concrete, concise steps. These steps may include combining data fields, elaborating on existing fields, or creating new fields derived from the existing data or excluding the data sample altogether if it is irrelevant. We generate this plan using an LLM, which is provided with the expanded task description, along with the optimized dataset, complete with its description and sample rows. A sample plan is shown in Figure 4. We are only required to create a plan once per task, since the same task-level plan will get executed for each row of the retrieved dataset. The Planning Module prompt is linked in the Appendix A.3.

Execution Module The execution of the transformation plan for each dataset sample is done using an LLM. The LLM receives three key pieces of information for each row of the dataset: the row itself, detailed specifications of the input task, and the transformation plan formulated earlier. Conditioned on these pieces of information, the model responds by producing an adjusted dataset row which, hopefully, meets the requirements of the input task. The Execution Module prompt is linked in Appendix A.4.

3.2 Using Multiple Datasets

The transformation process of data points may result in a significant number of them getting filtered out. Additionally many datasets may be small to begin with. Both these factors can result in in a reduced quantity of transformed data. To balance

Task Name	Task Category	Abbreviation	Task Instruction
Temporal Sequences	Logical Reasoning	Time	Answer questions about which times certain events could have occurred.
Code Line Descriptions	Coding	Code	Give an English language description of Python code.
Elementary Math	Math	Math	Answer a multiple choice mathematical word problem.
Cause and Effect	Causal Reasoning	C&E	Answer multiple-choice questions distinguishing cause and effect.
Medical Questions in Russian	Domain Specific	Russian	Answer a yes/no question about medical text in Russian.
Implicatures	Contextual QA	Impl.	Predict whether Speaker 2's answer to Speaker 1 is affirmative or negative.

Table 1: We evaluate our method on 6 diverse text-based tasks from BIG-Bench.

quantity and quality, we adopt a strategy of transforming multiple highly ranked datasets (up to four) till we reach our desired dataset set. Furthermore, if a considerable proportion of data sample transformations fail for a specific dataset, we opt to exclude it from consideration.

4 Experimental Setup

Evaluation procedure We evaluate DataTune and related baselines over 6 tasks from the BIG-Bench benchmark (Bench Authors, 2023), which we list in Table 1. BIG-Bench focuses on tasks believed to be beyond the capabilities of current language models, covering a variety of task categories. We choose tasks spanning logical reasoning, coding, math, causal reasoning, multilinguality, and domain specific tasks. For each task, we create 3000 data points through DataTune, and apply the training procedure highlighted earlier. We evaluate all models for two-shot performance, as per the BIG-Bench testing suite.[3]

Methods We compare the effectiveness of various **dataset collection** strategies, by using the resulting data to finetune the same base model, Mistral-7B (Jiang et al., 2023), a leading open-source LLM within its size category. As shown in Table 2, this includes: (1) retrieving existing data using dense retrieval (Viswanathan et al., 2023a), (2) generating synthetic data, (3) Prompt2Model (Viswanathan et al., 2023b) (a state-of-the-art method that combines retrieving existing data and synthetic data generation), (4) our DataTune approach, and (5) a combination of DataTune and synthetic data generation, which represents an integration of all the existing methods.

We also include two **prompting** baselines: (6) few-shot prompting on the base model Mistral-7B, and (7) `GPT-3.5-Turbo`, a significantly larger model, which we include as a robust benchmark due to its extensive capabilities.

Training Setup We used the Mistral-7B model, following the approach of Jiang et al. (2023), and applied the QLoRA technique from Dettmers et al. (2023) for fine-tuning. The process was carried out over 3 epochs. We select parameters by running 4 runs across two sets of hyperparameters over two values(learning rate: $5e^{-5}$ and $1e^{-4}$, and QLoRA's α parameter between 16 and 24 and choose the run with the lowest validation loss at any point. We used the AdamW optimizer (Loshchilov and Hutter, 2017) and set QLoRA $r = 8$. We conduct our training on 2 NVIDIA RTX A6000 GPUs.

Metrics Following Bench Authors (2023), we use a *normalized aggregate score* evaluation that normalizes a multiple choice grade score, such that a score of 0 implies random chance performance, and 100 implies human expert performance. Additionally, a score less than 0 indicates performance worse than random chance.

5 Results and Analysis

5.1 Performance Comparison

DataTune consistently outperforms few-shot prompting, as well as existing individual data collection methods. From Table 2, we see that fine-tuning our base model on DataTune outperforms the base Mistral-7B model by 6.4 points on average, improving over it in five out of six tasks. We also show that DataTune provides an average improvement of 11 and 2.9 points over fine-tuning on existing data and synthetically generated data respectively.

DataTune's transformation is complementary to synthetic data generation. It is noteworthy that the combination of DataTune and synthetically gener-

[3] https://github.com/google/BIG-bench

	Method	Steps		Tasks						
		Retrieval Type	Generation	Time	Code	Math	C&E	Russian	Impl.	**Avg.**
	GPT-3.5 (few-shot)	-	-	50.6	75.6	30.4	96.7	90.6	64.2	68.0
	Mistral-7B (few-shot)	-	-	-2.5	62.3	2.9	37.2	39.8	39.0	29.8
Mistral-7B+	Existing data	Dense	-	-4.7	62.3	0.8	52.9	0.0	39.9	25.2
Mistral-7B+	Synthetic data	-	Synthetic	2.0	60.8	3.8	37.2	54.0	41.9	33.3
Mistral-7B+	DataTune	+ Reranker	Transformed	-2.1	71.2	1.3	**56.9**	48.0	41.9	36.2
Mistral-7B+	Prompt2Model	Dense	Synthetic	-2.0	73.4	4.7	33.8	**86.0**	44.0	40.0
Mistral-7B+	DataTune+Synthetic	+ Reranker	Both	**16.9**	**84.5**	**8.1**	41.2	68.0	**48.0**	**44.5**

Table 2: We compare the performance of different few-shot learning methods across six BIG-Bench tasks. Here, we categorize each method by what base model is used (Mistral-7B or GPT-3.5-Turbo), whether data is retrieved (and, if so, whether a dense retriever or dense retriever + reranker is used), how the data points are generated (whether transformed from an existing dataset, generated synthetically, or both). Normalized aggregate scores below zero imply performance worse than chance.

ated data results in a marked performance increase. This synergistic improvement yields an overall average score of 44.5 on the BIG-Bench tasks we consider. We provide a more detailed analysis of the synergy between DataTune and existing synthetic dataset generation methods in Section 5.5.

Our system (DataTune + Synthetic Data Generation) outperforms SOTA baselines like Prompt2Model. In order to make a fair comparison with baselines such as Prompt2Model that fine-tune on both existing data and synthetically generated data, we define our system as the base Mistral-7B model fine-tuned on DataTune-created data and synthetically generated data. Our analysis in Table 2 highlights significant differences in performance between our system and Prompt2Model, with our system demonstrating a notable advantage over Prompt2Model five out of six tasks, with an average improvement of 8.3 points. These findings underscore the effectiveness of our system to create quality datasets for fine-tuning across a wide range of tasks and domains.

5.2 DataTune Improves Dataset Diversity

Sampling diverse yet high-quality examples from a language model is a challenging task. Prior work has shown that the correctness of a synthetic dataset sampled from a language model is inversely correlated with its diversity (Ye et al., 2022). We find that, since dataset transformation conditions on a retrieved dataset, the data generated by DataTune can be much more diverse than the data generated by existing synthetic dataset generation methods.

Generating synthetic data directly from a language model (using a method like Prompt2Model) often contains near-duplicate examples. Does transforming existing datasets reduce the incidence of duplicate examples? Using ROUGE-L to determine lexical uniqueness (Lin, 2004; Wang et al., 2022), we determine a sentence s in dataset D to be *unique* up to threshold[4] T if $\max_{s' \in D \setminus \{s\}} Rouge(s, s') < T$. In Figure 5, we observe that over 50% of examples generated synthetically are near-duplicates for 3 of 5 tasks; in each of those cases, using dataset transformation instead effectively eliminates this problem.

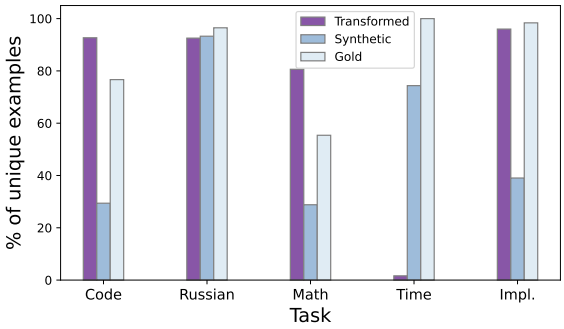

Figure 5: Synthetic dataset generation often suffers from the problem of generating multiple duplicates of the same example in a given dataset. On 3 of 5 tasks, we find that data transformation from retrieved datasets significantly mitigates this issue. The other two datasets, Russian and Temporal, represent failure modes of our system. Gold represents the BigBench Dataset for a given task.

We observe similar trends with lexical diversity across all dataset creation methods. In Table 3, we measure the number of unique bigrams per generated input example. Dataset transformation significantly increases the number unique bigrams per

[4]We set the ROUGE-L threshold to 0.8 for Code Line Descriptions, where examples are Python snippets, 0.9 for Temporal Sequences, where examples are long English texts, and 0.7 for the other datasets.

Dataset	Unique Bigrams Per Example	Total Tokens Per Example
Code Line Description		
Gold	13.2	32.3
Synthetic	2.5	35.0
Transformed	14.9	86.9
Elementary Math		
Gold	10.8	48.6
Synthetic	3.3	34.4
Transformed	11.6	43.8
Implicatures		
Gold	9.9	24.1
Synthetic	2.7	27.7
Transformed	17.8	39.8
Temporal Sequences		
Gold	1.0	99.7
Synthetic	20.8	54.6
Transformed	0.2	73.7
Medical Questions in Russian		
Gold	62.0	79.4
Synthetic	20.8	54.6
Transformed	11.6	44.8

Table 3: We observe that dataset transformation yields datasets with greater lexical diversity than synthetic dataset generation on 3 of 5 datasets.

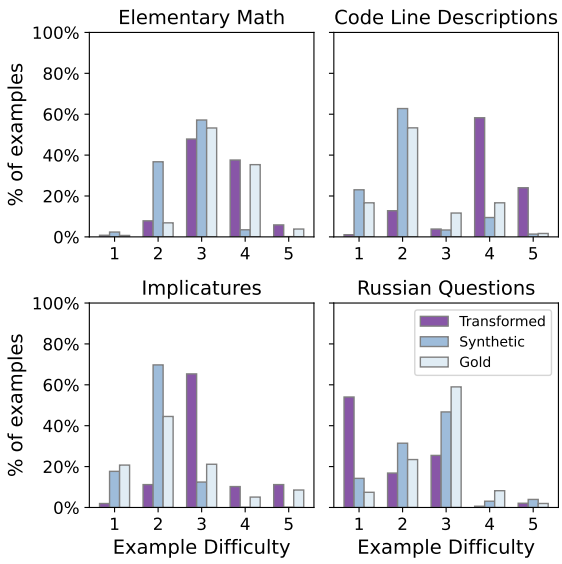

Figure 6: Dataset Transformation leads to more difficult examples than synthetically generated examples, which are over-represented by easy examples, relative to manually-curated BIG-Bench evaluation datasets (Gold).

example (and moderately increases the length of each example) on 3 of 5 datasets.

According to both measures, dataset diversity decreases on *Temporal Sequences* and *Medical Questions in Russian*, which are also the two tasks where DataTune fails to improve over training on fully synthetic data. We discuss these two tasks in Section 5.5 and Limitations, respectively.

5.3 DataTune Generates Harder Examples

Synthetic datasets sampled directly from a language model tend to be overrepresented with easy-to-solve examples (Xu et al., 2023). To test our ability to overcome this issue, we used `gpt-3.5-turbo` to estimate the difficulty of each example generated from each method (DataTune, synthetic generation, and manual curation) for four tasks. We wrote a custom difficulty estimation prompt (including in-context examples) for each task; in all prompts, we specify that the LLM should rate the example on a scale of 1 to 5. As illustrated in Figure 6, we observe that the data from DataTune exhibits a higher level of difficulty compared to synthetically generated data for all datasets other than *Medical Questions in Russian*. We attribute this increase in difficulty to the added knowledge introduced by the existing dataset selected for transformation, which is often more nuanced and, consequently, more challenging. In contrast, unconditional generation from LLMs tends to produce examples that are likely to have a high prior (prompt-independent) probability under the language model. The exception to this effect, *Medical Question in Russian*, can be attributed to the failure of our approach's transformation plan to generate data in the correct language for this task. We provide more details of this failure mode in Limitations.

5.4 Does Generating Harder Examples Lead To Lower Label Accuracy?

We evaluated the label accuracy of generated data for the Code Line Description task by comparing DataTune against synthetic dataset generation. For this purpose, we manually annotated a random sample of 100 data points from each generation method for the Code Line Description Task. We found that the label accuracy from DataTune is 86%, compared to 88% for synthetic data generation. This comparison suggests that DataTune can produce datasets that are comparable in accuracy to purely-synthetic datasets, despite DataTune generating significantly more diverse and challenging examples on this task.

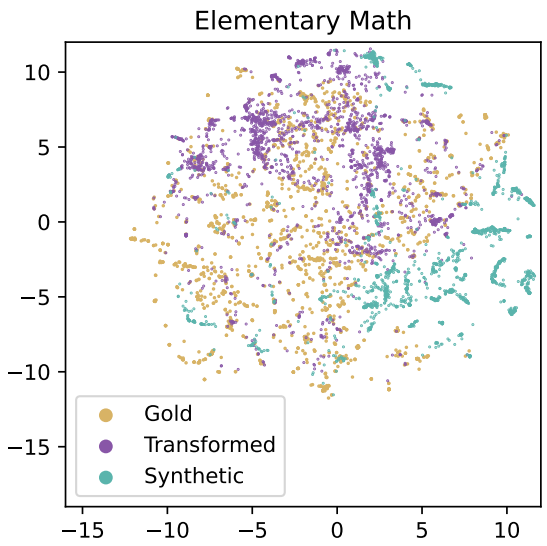

Figure 7: We observe that examples generated via DataTune and synthetic dataset generation fall into visually well-separated regions of embedding space, after projecting to two dimensions via t-SNE.

5.5 Transformed Data Can Be Complementary to Synthetic Data

In two cases where DataTune fails to improve over a synthetic data baseline, *Temporal Sequences* and *Elementary Math*, our combined DataTune + Synthetic system still outperforms all other comparable (Mistral-based) baselines. We observe these two approaches to dataset generation can be complementary to each other, yielding additive improvements when combined.

For a concrete example of this, we can visually observe the two-dimensional semantic distribution of questions generated for the Elementary Math task via DataTune, Synthetic Generation, and from the gold dataset in Figure 7. We encoded each question using MiniLM v2 (Wang et al., 2021) via `sentence-transformers` (Reimers and Gurevych, 2019), then projected each embedding into a 2D space using t-SNE (van der Maaten and Hinton, 2008).

Visually, we observe that the embedding clusters of questions generated via DataTune and Synthetic Generation appear to be largely disjoint. This supports our hypothesis that these two methods systematically produce semantically different examples, and therefore combining the examples from each method can lead to a synergistic effect.

6 Conclusion and Future Work

We present DataTune, an improved approach for enhancing automatic dataset generation by transforming existing labeled datasets. Our method significantly outperforms existing automatic dataset generation techniques on several challenging tasks from BIG-Bench. When used in conjunction with existing synthetic dataset generation methods, it achieve superior performance compared to other few-shot learning methods on the base model we used (Mistral-7B). Our analysis reveals that DataTune not only creates more diverse and accurate datasets but also increases their complexity for fine-tuning. An important direction is whether transformation-based dataset creation methods like DataTune can still be effective when examples are retrieved from the open web rather than from the collections of manually-curated datasets we consider in our work. Another important direction for future work will be to generate code to execute a transformation plan (rather than querying an LLM for each instance of data). Both of these future directions would improve the accessibility and scalability of our suggested approach.

Limitations

We identify four key limitations in our system:

1. **LLM Query Cost**: Our dataset transformation component requires running the Execution Module on each row of each dataset we wish to transform. Given that our Execution Module prompts a large language model, the number of LLM queries scales linearly with the amount of data to be transformed. This could be cost-prohibitive for transforming very large datasets. This LLM usage requirement could also exclude members of the research community without consistent LLM access from benefiting from our work.

2. **Dependence on the Planning Module**: Our dataset transformation system relies heavily on the Planning Module to produce clear and comprehensive instructions for the Execution Module to enact. Our Planning Module operates by prompting a large language model. Given that prompted LLMs can behave unpredictably in changes in prompts (Sclar et al., 2023), an important limitation of our method is that its success depends on a large language

model following a prompt correctly. We see this failure with Medical Questions in Russian, where the agent is supposed to translate a Russian dataset into English to facilitate the generation of question-answer pairs. When running our benchmarking, the Planning Agent failed to translate the generated question-answer pairs back to Russian. This resulted in a training dataset that, despite being conceptually close, was practically far from correct.

3. **Handling Non-English Data**: Our transformation agent's ability to process non-English data tasks is substantially compromised, frequently altering example tasks instead of the actual data. This deficiency is primarily due to the reliance on models like GPT 3.5, which have been extensively trained on English data, thereby diminishing their proficiency with other languages.

4. **Dependence on Instruction-Following LLMs**: The system's execution component depends on Large Language Models (LLMs) that are specifically designed to adhere to instructions. We have identified discrepancies in performance among LLMs tailored for instruction-based tasks versus those developed for conversational purposes. This limitation confines our system to using only a narrow selection of LLMs that demonstrate the ability to follow instructions accurately.

Ethical Considerations

DataTune could make it easier for the general public to build custom language models. The broadening of open-ended technology induces ethical concerns, similar to the issues with open-source deepfake libraries described by Widder et al. (2022). While DataTune has potential for misuse, this is likely no greater than the potential harms presented by the underlying open-source large language models. By making it easier to build task-specific language models, we hope that these risks are balanced by the benefits of making NLP models accessible to those outside the NLP community or those without the resources to manually collect labeled data. We aim to be transparent in our documentation about the potential limitations of the system.

The platform is particularly valuable for individuals outside the NLP community who can benefit from using NLP models in their work but lack the specialized knowledge to develop these tools independently. The decision to open-source DataTune invites community contributions, emphasizing a collaborative approach to improve and expand the tool's capabilities. This strategy not only enhances the system's utility but also aligns with a broader goal of increasing the accessibility of NLP innovations and fostering a more inclusive technological environment.

References

BIG Bench Authors. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Chris Callison-Burch. 2009. Fast, cheap, and creative: Evaluating translation quality using amazon's mechanical turk. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, pages 286–295.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Bosheng Ding, Chengwei Qin, Linlin Liu, Yew Ken Chia, Boyang Li, Shafiq Joty, and Lidong Bing. 2023. Is GPT-3 a good data annotator? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11173–11195, Toronto, Canada. Association for Computational Linguistics.

Olivia Huang, Eve Fleisig, and Dan Klein. 2023. Incorporating worker perspectives into MTurk annotation practices for NLP. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1010–1028, Singapore. Association for Computational Linguistics.

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

A van Lamsweerde. 2009. *Requirements engineering: from system goals to UML models to software specifications*. John Wiley & Sons, Ltd.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario vSavsko, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp

Schmid, Sylvain Gugger, Clement Delangue, Th'eo Matussiere, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. *ArXiv*, abs/2109.02846.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12284–12314, Toronto, Canada. Association for Computational Linguistics.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. 2023. Is chatgpt good at search? investigating large language models as re-ranking agent. *ArXiv*, abs/2304.09542.

Ruixiang Tang, Xiaotian Han, Xiaoqian Jiang, and Xia Hu. 2023. Does synthetic data generation of llms help clinical text mining?

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Vijay Viswanathan, Luyu Gao, Tongshuang Wu, Pengfei Liu, and Graham Neubig. 2023a. DataFinder: Scientific dataset recommendation from natural language descriptions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10288–10303, Toronto, Canada. Association for Computational Linguistics.

Vijay Viswanathan, Chenyang Zhao, Amanda Bertsch, Tongshuang Wu, and Graham Neubig. 2023b. Prompt2model: Generating deployable models from natural language instructions.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. 2020. Exploring and predicting transferability across nlp tasks. *arXiv preprint arXiv:2005.00770*.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2140–2151, Online. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

David Gray Widder, Dawn Nafus, Laura Dabbish, and James Herbsleb. 2022. Limits and possibilities for "ethical ai" in open source: A study of deepfakes. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, page 2035–2046, New York, NY, USA. Association for Computing Machinery.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions.

Jiacheng Ye, Jiahui Gao, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. 2022. Progen: Progressive zero-shot dataset generation via in-context feedback. In *Conference on Empirical Methods in Natural Language Processing*.

Zhisong Zhang, Emma Strubell, and Eduard Hovy. 2022. A survey of active learning for natural language processing. *arXiv preprint arXiv:2210.10109*.

A Example Appendix

A.1 Reranking Prompt

Your objective is to choose the most relevant dataset for a given a task (and few examples of the task). For each dataset, you will be provided with the dataset description, and tags related to the dataset. Please return the most relevant dataset, e.g., squad
The following is the task
{{instruction}}
and these are some examples of the same:
{{examples}}
There are {{num}} datasets available for this task.
{{datasets}}
The name of the most relevant dataset for this task is:

where each dataset in {{datasets}} is defined as:

{counter}{{dataset_name}}:Description-{{dataset_description}}.
This dataset has the following tags:
{{tags}}

A.2 Schema Selection Prompt

Your objective is to carefully analyze the task and the dataset mentioned, and decide whether the columns are relevant input, relevant output, irrelevant for the given task, or if it is ambiguous. There should be at most one output column. It is possible to have no relevant columns, in which case return the input and output column as empty lists. Answer in a json format, with the following keys: input, output, irrelevant, ambiguous.
{{INCONTEXT_EXAMPLES}}
After seeing these examples with the required columns, please provide the relevant columns for this context:

You are tasked with the following process. {{instruction}} For this task, you will use the {{dataset_name}} dataset from HuggingFace. Dataset Description: {{dataset_description}}
A sample data instance from this is as follows. {{sample_row}}.
This dataset has the following columns: {{dataset_columns}}
Required Columns :

A.3 Planning Module Prompt

You are a Planning Agent. You create a plan to transform data samples from their existing format into the required format for a given task.

Here are some examples for your reference.
{{in_context_examples}}

Now do the following task:
Task Description: {{task_description}}
Task Examples: example
Here are samples from a potentially relevant dataset for the task above. Notice how the format below is not as required by the task above.
Dataset Samples: {{dataset_row}}
Carefully analyze the 'Task Description' and the 'Task Examples'. Propose a higher-level plan to convert data from the Dataset Sample to data in the required format task examples. Your plan should be a list of sequential steps that can be taken to perform the data transformation. You don't need to use all columns, as the dataset may not be fully relevant. Keep steps as simple, explicit and concise as possible. Each step in the plan may take any of the following actions: 1. Generate new columns as required by the task, and save them
2. Expand on a particular column to make it something more relevant to the task and save it
3. Combine multiple columns from the dataset
4. Choose columns that will form "input"
5. After the input field is created, carefully analyze it to choose/generate the output field
6. Ignore a data sample because it is not all relevant and return null for them.

Return only the plan.

A.4 Execution Module Prompt

You are a Data Transforming Agent. Your job is to transform data from a given format to the required format. Following are the detailed instructions for the same: 1. Read the 'Task Description'.
2. An example of the input and output looks like for the task is shown in 'Task Examples'
3. The sample to be transformed is in 'Data Sample'.
4. Read the data transformation plan carefully that will help you convert the 'Data Sample' into the required format. This should be relevant and intune to the 'Task Description'
5. Perform the plan step by step and explain your thinking.
6. End your response with the transformed sample as a JSON response with exactly 2 fields: "input" and "output".

Here are some examples for your reference. {{incontext_examples}}

Now do the following task:
Task Description: {{task_description}}
Task Examples: {{sample}}
{{plan}}
Dataset Sample: {{dataset_row}}
Think step by step through the plan to convert the above 'Dataset Sample' and show your working. End your response as a JSON with exactly two fields: "input", and "output" Response: