FABRICATOR: An Open Source Toolkit for Generating Labeled Training Data with Teacher LLMs

Jonas Golde[1], **Patrick Haller**[1], **Felix Hamborg**[1], **Julian Risch**[2], **Alan Akbik**[1]
[1] Humboldt University of Berlin
[2] deepset GmbH
{jonas.golde, patrick.haller.1, felix.hamborg, alan.akbik}@hu-berlin.de
julian.risch@deepset.ai

Abstract

Most NLP tasks are modeled as supervised learning and thus require labeled training data to train effective models. However, manually producing such data at sufficient quality and quantity is known to be costly and time-intensive. Current research addresses this bottleneck by exploring a novel paradigm called *zero-shot learning via dataset generation*. Here, a powerful LLM is prompted with a task description to generate labeled data that can be used to train a downstream NLP model. For instance, an LLM might be prompted to "*generate 500 movie reviews with positive overall sentiment, and another 500 with negative sentiment*." The generated data could then be used to train a binary sentiment classifier, effectively leveraging an LLM as a teacher to a smaller student model. With this demo, we introduce FABRICATOR, an open-source Python toolkit for dataset generation. FABRICATOR implements common dataset generation workflows, supports a wide range of downstream NLP tasks (such as text classification, question answering, and entity recognition), and is integrated with well-known libraries to facilitate quick experimentation. With FABRICATOR, we aim to support researchers in conducting reproducible dataset generation experiments using LLMs and help practitioners apply this approach to train models for downstream tasks.

1 Introduction

In recent years, natural language processing (NLP) has witnessed remarkable progress due to the introduction of pre-trained language models (PLMs) (Devlin et al., 2019; Liu et al., 2019; Conneau and Lample, 2019; He et al., 2021). These PLMs are typically fine-tuned on large human-annotated datasets, resulting in state-of-the-art performance in tasks such as text classification, token classification, and question answering. However, real-world applications of this approach face the bottleneck that sufficient amounts of human-annotated data are often unavailable and too costly to produce manually, especially when domain expertise is required.

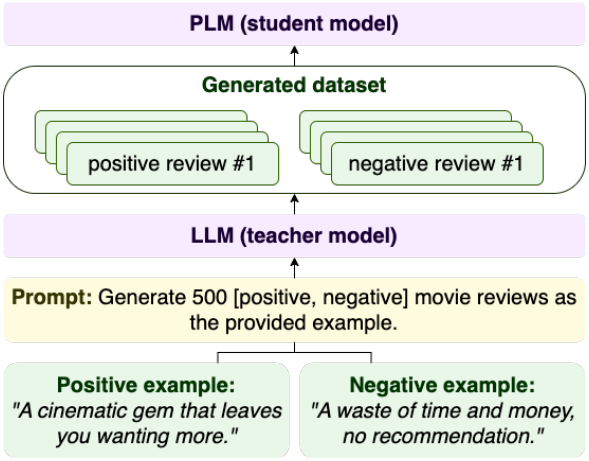

Figure 1: The process of *learning via dataset generation*. A teacher model (LLM) is prompted to generate 500 movie reviews for each sentiment (positive, negative). A smaller student PLM is trained on the generated dataset.

Dataset generation with teacher LLMs. Recently, a paradigm called *zero-shot learning via dataset generation* (Meng et al., 2022; Ye et al., 2022a,b) has emerged, potentially obviating the need for human-annotated data. This approach leverages the generation capability of large language models (LLMs) to create class-conditioned texts guided by label-descriptive prompts and, optionally, *few-shot* examples of instances of the desired classes. The generated dataset is then used to train a smaller student PLM.

Refer to Figure 1 for an illustration of this process: In this example, an LLM is instructed to write 500 positive and 500 negative movie reviews. To guide the process, we include an example of a positive and negative review in the prompt. With this

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 1–11

prompt and 1-shot example, we generate a dataset of 1,000 movie reviews labeled with binary sentiment. This dataset is used to train a student model to perform binary sentiment analysis.

Limitations. However, despite the conceptual simplicity of using LLMs to generate training data, many open questions remain regarding the specifics and ultimate potential of this approach. Questions include: (*1*) How to best prompt the LLM and whether to include examples in the prompt, (*2*) For which downstream NLP task families and specific tasks this approach is effective, and (*3*) Whether it is better to generate large amounts of training data or focus on smaller, high-quality generation efforts. While various current works are investigating these questions for specific tasks, we find that, at present, no open-source library specifically supports research on dataset generation with LLMs.

Contributions. To close this gap, we present FABRICATOR, an open-source Python library for dataset generation with LLMs. Our main goals are to facilitate experimentation, enable the application of dataset generation to specific downstream tasks, and encourage the reproducibility of experiments.

FABRICATOR modularizes the dataset generation process and provides a simple interface to facilitate experimentation: Users may choose which LLM to use, define prompts and label definitions, and leverage existing NLP datasets for few-shot examples and NLP task definitions. Our library includes an integration into HuggingFace's DATASETS library (Lhoest et al., 2021), allowing users to easily share generated datasets and use them for training NLP models. We provide examples for various NLP task families, including text classification, textual entailment, question answering, and entity recognition. In this paper:

- We introduce the FABRICATOR library, give an overview of core concepts and usage workflows (Section 2).
- We present a set of example experiments in which FABRICATOR is used to create datasets for various text classification, question answering, and textual entailment tasks (Section 3).

We publish the code on GitHub[1] under the Apache 2 license.

[1] https://github.com/flairNLP/fabricator

2 FABRICATOR

We first give a high-level overview of supported generation workflows in FABRICATOR (Section 2.1), discuss the main classes and concepts (Section 2.2), and walk through an example use case and script (Section 2.3).

2.1 Generation Workflows

Depending on the downstream task, researchers may have one of three data generation targets we support in FABRICATOR:

1. Generate unlabeled data. The first generation target is to *produce unlabeled data*. For instance, during the development of a question answering system, we might require a corpus of example questions or a corpus of texts on a particular topic. For this scenario, users provide a prompt $\mathbf{w}$ (such as "*Generate a text in the domain of history that contains facts someone can ask questions about.*"), and the auto-regressive LLM G_θ generates appropriate text $\mathbf{x}^g$.

2. Generate label-conditioned data. The second generation target is generating data belonging to a pre-defined class, such as classification tasks. The LLM generates a text $\mathbf{x}^g$ corresponding to a specific label y from a set of labels.

As discussed in the introduction, one example is to generate training data for a binary sentiment classifier. To achieve this, one must define a set of labels ($\mathbf{y} = \{positive, negative\}$) and a prompt $\mathbf{w}_y$ such as "*Generate a <y> movie review:*." The generated sequence $\mathbf{x}^g$ will be paired with the label y to form a training pair $(\mathbf{x}^g, y)$ for fine-tuning.

3. Annotate unlabeled data. The third generation target holds if an unlabeled text dataset for a domain is already available and only training labels are missing. For instance, a corpus of movie reviews might already be available, but sentiment labels are missing.

In FABRICATOR, researchers can add labels to an existing corpus by extending prompt $\mathbf{w}$ with fixed label options $\mathbf{y}$ to form $\mathbf{w_y}$ like "*Annotate the movie review either as: positive, negative.*" The generated label y is then paired with the unlabeled data point $\mathbf{x}^u$ to form a data pair $(\mathbf{x}^u, y)$.

The generation targets defined above will be executed multiple times to generate a corpus of a specified size. The prompt may also be extended to

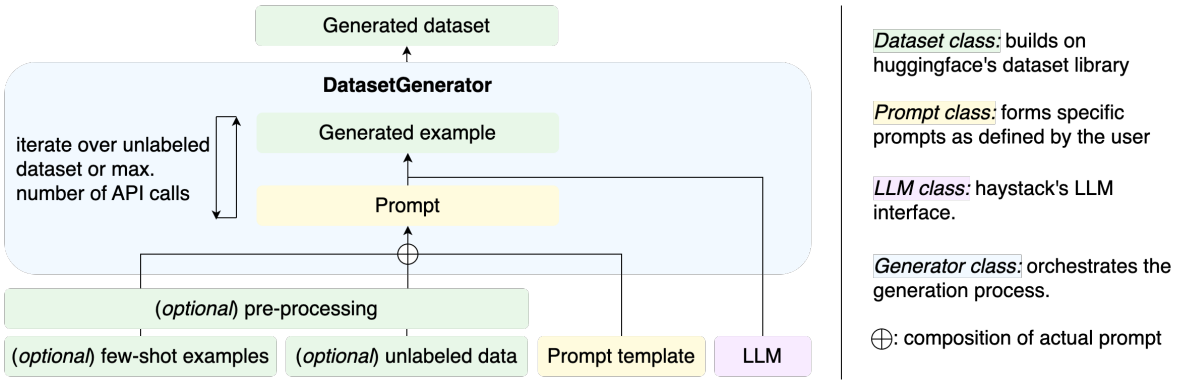

Figure 2: With FABRICATOR, the generation process involves a prompt template that creates the final prompt using all provided arguments. The generator class creates training examples until the maximum number of prompt calls is reached, or the unlabeled dataset is fully annotated. Ultimately, the generator class produces a HuggingFace `Dataset` instance.

include few-shot examples of each class, as shown in Figure 1. The prompt can also handle multiple inputs (for example, for tasks like textual similarity) using pre-defined interfaces in FABRICATOR. In all cases, the correct prompt is composed and executed in our backend.

2.2 Classes and Concepts

As Figure 2 illustrates, the key module in our approach is the `DatasetGenerator` class, which acts as an orchestrator between the LLM (`PromptNode`), the prompt (`BasePrompt`), and optionally, the few-shot examples and unlabeled datasets.

The `generate()` function within the `DatasetGenerator` class converts the `BasePrompt` and the provided few-shot and unlabeled data into a processable prompt for the LLM. The method offers various arguments to steer the generation process. Users can specify parameters like the maximum number of API calls, the sampling strategy of few-shot examples (uniform vs. stratified), or the number of few-shot examples to use in a single prompt. Our repository contains documentation with details on all available customization options.

2.2.1 HuggingFace Interoperability through Dataset Class

FABRICATOR operates on the `Dataset` class from HuggingFace's DATASETS library. By default, `generate()` produces the generated data as a `Dataset` instance. This allows generated datasets to be directly used in existing training scripts of the TRANSFORMERS library (Wolf et al., 2020) and to be shared among researchers via the Huggingface dataset hub.

An existing dataset may also be used as input to the `generate()` method. Since the DATASETS library supports a wide range of standard benchmarks and their formats, existing datasets can be easily loaded and used as input. For instance, in some generation workflows, we would like to add labels to an existing corpus or use instances as few-shot examples within a prompt.

2.2.2 Prompt Class

Prompting is crucial when operating on large language models as it guides the auto-regressive generation process. While in the simplest case, a prompt is a single textual string, we find that many scenarios require more complex prompts and customization options. For instance, when including few-shot examples in a prompt, questions include how many examples to include in each prompt and how these are sampled (uniform vs. stratified) from available few-shot data across different prompt calls. Similarly, the complexity increases for tasks such as textual entailment (requiring multiple inputs) and entity recognition (potentially requiring transformation of token-level BIOES tags into span-level prompting queries).

To address these challenges, FABRICATOR introduces a simple yet powerful `BasePrompt` class that offers clear interfaces for customizing prompts for various dataset generation tasks. The interface includes attributes to specify pre-defined label options for label-conditioned generation, and support for having few-shot examples or unlabeled datasets by selecting the relevant columns for generation and few-shot information in the prompt.

Since the prompt class directly operates on the dataset columns, FABRICATOR enables a sophis-

```python
import os
from datasets import load_dataset
from haystack.nodes import PromptNode
from fabricator import DatasetGenerator, BasePrompt

dataset = load_dataset("processed_fewshot_imdb", split="train")

prompt = BasePrompt(
    task_description="Generate a {} movie review.",
    label_options=["positive", "negative"],
    generate_data_for_column="text",
)

prompt_node = PromptNode(
    model_name_or_path="gpt-3.5-turbo",
    api_key=os.environ.get("OPENAI_API_KEY"),
    max_length=100,
)

generator = DatasetGenerator(prompt_node)
generated_dataset = generator.generate(
    prompt_template=prompt,
    fewshot_dataset=dataset,
    fewshot_sampling_strategy="uniform",
    fewshot_examples_per_class=1,
    fewshot_sampling_column="label",
)
generated_dataset.push_to_hub("generated-movie-reviews")
```

Listing 1: A script that uses FABRICATOR and generates additional movie reviews based on few-shot examples.

ticated and flexible prompt design. To illustrate, when performing a textual similarity task, the user can specify the first sentence and the label as the few-shot information and prompt the LLM to generate a second sentence corresponding to the given sentence and label.

2.2.3 LLMs

The LLM interface must be stable and ideally compatible with models hosted as APIs or self-hosted LLMs. We leverage the HAYSTACK[2] framework (Pietsch et al., 2019), specifically the `PromptNode` class, for interactions with LLMs. The `PromptNode` implementation allows users to select and use LLMs from various model providers, including HuggingFace, OpenAI, Azure, Anthropic, and Cohere.

2.3 Example Script

In Listing 1, we introduce an example script in which FABRICATOR is used to generate additional movie reviews for training a binary sentiment classification model (refer to generation workflow 2 as defined in Section 2.1). To implement this, we define:

- a pre-processed few-shot dataset (`dataset`, line 6) having labels in natural language form (e.g., 0 becomes "negative"). These examples are used to augment the generation prompt,
- a prompt template (`prompt`, line 8) specifying the instruction to the LLM,
- an LLM to use as teacher model (`prompt_node`, line 14),
- a `DatasetGenerator` to execute the generation process with all parameters (`generator`, line 20).

The prompt is configured in the constructor of the `BasePrompt` class (lines 8-12): We set a `task_description` with a placeholder for `label_options` that we provide as a separate argument. We also specify for which column in the loaded dataset to predict labels.

We then define a teacher LLM (lines 14-18) and pass datasets, prompt, and LLM to the `DatasetGenerator` orchestrator class (lines 20-27). Here, we specify a few-shot strategy to sample one label from the "label" column uniformly during generation. We do so to generate either a positive or a negative review. Upon completion, the `generate` function returns the annotated `Dataset` instance.

[2] https://github.com/deepset-ai/Haystack

Dataset	Labels	# Training examples			
		50	500	1k	all (max. 10k)
IMDB	Gold	37.6 ± 35.8	88.5 ± 0.8	90.0 ± 0.4	93.0 ± 0.2
	Generated	**53.8** ± 11.5	**88.8** ± 0.6	**90.2** ± 0.4	**92.0** ± 0.1
MRPC	Gold	66.6 ± 0.8	73.0 ± 1.3	75.2 ± 1.1	83.9 ± 0.2
	Generated	**68.4** ± 0.8	**72.1** ± 1.0	72.4 ± 1.2	75.8 ± 0.7
SNLI	Gold	38.5 ± 2.5	64.7 ± 0.9	71.3 ± 0.7	82.1 ± 0.4
	Generated	**42.2** ± 2.4	54.8 ± 1.0	56.1 ± 1.1	63.1 ± 0.7
TREC-6	Gold	50.4 ± 7.6	93.6 ± 0.6	94.9 ± 1.1	97.5 ± 0.4
	Generated	39.8 ± 4.5	79.3 ± 2.2	80.8 ± 3.0	82.4 ± 1.1
SQuAD	Gold	-	-	39.1 ± 4.9	68.8 ± 0.5
	Generated	-	-	**46.8** ± 1.1	52.5 ± 0.3

Table 1: Results on re-annotation experiments using 2 few-shot examples per prompt (uniformly sampled from 6 few-shot examples per class). We report accuracy except for SQuAD, where we report F1, and highlight bold those experiments where generated data yielded similar scores as human-annotated data. We observe that GPT-3.5 is not able to annotate on human-level performance except for simple classification tasks such as IMDB.

3 Experiments

To illustrate how FABRICATOR could be used in research, we conduct an exploratory evaluation of two scenarios: (*1*) how models trained on generated datasets compare to models trained on human-annotated datasets, and (*2*) whether few-shot examples in the prompt improve generated datasets.

To do so, we train smaller PLMs on generated datasets and evaluate them on the human-labeled test split of the respective benchmark. For question answering, we fine-tune a `roberta-base` PLM (Liu et al., 2019). For all other tasks, we fine-tune a `bert-base-uncased` PLM (Devlin et al., 2019). The hyperparameters are listed in Appendix A.2. We report the score and standard deviation averaged over 5 random seeds for each experiment.

3.1 Experiment 1: Comparison of Generated and Human-Annotated Datasets

We re-annotate existing benchmark datasets with generated labels in the first experiment. This experiment aims to measure the difference in accuracy of downstream task models trained on human-annotated data compared to models trained on generated data. We evaluate text classification, textual similarity, and extractive question answering tasks.

Experimental setup. We conduct this evaluation on 5 datasets spanning 3 NLP tasks: We use IMDB (Maas et al., 2011), a binary sentiment classification benchmark, and TREC-6 (Li and Roth, 2002), a 6-class question type categorization dataset to evaluate text classification tasks. We use the 2-class MRPC (Dolan and Brockett, 2005) and the 3-class SNLI (Bowman et al., 2015)) datasets to evaluate textual similarity tasks. Finally, we use SQuAD-v2 (Rajpurkar et al., 2016)) to evaluate extractive question answering. We use generation prompts augmented by 2 examples per prompt sampled from 6 possible few-shot examples per class.

Results (Table 1). For all datasets, we compare a generated dataset of 50, 500, 1k and the full dataset (limited to 10k if it is larger) to gold-annotated data of the same size. For question answering, models need to be trained on at least 1k to obtain representative results, so we do not report scores for 50 or 500 examples for SQuAD.

We find that for simple tasks such as binary sentiment classification (IMDB), models trained on the annotations by LLMs achieve similar accuracy on the gold-labeled test split (↓1.0 pp. in accuracy with 10k training examples). However, we as the complexity of datasets increases (text classification with more classes and extractive question answering), we observe that the performance of models trained on LLM-annotated datasets falls short (↓19.0 pp. for SNLI and ↓16.3 pp. for SQuAD, with 10k training examples).

These performance gaps indicate that the usefulness of LLMs as teacher models depends on the specific task. In the next section, we present an experiment that explores how to close this gap by using additional few-shot examples.

Dataset	# few-shot examples per class	# examples per class used in prompt				
		0	1	2	3	4
TREC-6	0	45.5 ± 2.3	-	-	-	-
	2	-	70.0 ± 1.6	65.5 ± 0.9	-	-
	4	-	79.5 ± 1.1	71.1 ± 2.0	**86.6** ± 0.6	69.8 ± 1.5
	8	-	76.1 ± 1.9	**79.5** ± 1.3	**81.0** ± 1.8	**87.4** ± 0.6
	16	-	72.7 ± 2.1	78.1 ± 1.9	**81.0** ± 2.4	74.2 ± 1.4

Table 2: Results on 500 annotated TREC-6 examples using varying amounts of few-shot examples. We sweep over the number of few-shot examples and the number of few-shot examples used in the actual prompt. We highlight bold where increasing few-shot examples improves over the 79.3 TREC-6 score of Experiment 1 (Table 1).

3.2 Experiment 2: Impact of Few-Shot Examples

In the second example experiment, we re-annotate TREC-6 using a varying number of few-shot examples. This experiment aims to determine whether adding few-shot examples for each class improves dataset generation with FABRICATOR. We investigate two variables: (*1*) The total number of available few-shot examples per class and (*2*) the actual number of few-shot examples included per prompt. For instance, there might be 8 few-shot examples available in total, but only 3 are randomly sampled to be included in each prompt call.

Results (Table 2). We note a generally positive trend in that increasing the number of available few-shot examples (column *# few-shot examples per class*) and increasing the number of examples used in each prompt (column *# examples per class used in prompt*) improves model performance. In particular, we find many settings that outperform the numbers of our previous experiment (where we sampled 2 examples per prompt out of a total of 6 possible examples), highlighted bold in Table 2.

However, we also find that improvements become uneven when *# examples per class used in prompt* is increased above 3, indicating prompts should not be overloaded with too many examples.

4 Related Work

Significant progress has been achieved in enhancing dataset generation with teacher LLMs (Schick and Schütze, 2021b; Meng et al., 2022; Ye et al., 2022a; Bonifacio et al., 2022; Peng et al., 2023; Meng et al., 2023), effectively selecting few-shot examples (Liu et al., 2022; Gunasekar et al., 2023) and assessing the quality of datasets produced by LLMs (Gilardi et al., 2023; Chen et al., 2023). However, we note a lack of accessible frameworks that facilitate straightforward and reproducible dataset generation using teacher LLMs. While existing open-source toolkits like OpenPrompt (Ding et al., 2022) partially extend to dataset generation scenarios, our approach stands apart by having lightweight, dedicated interfaces for the introduced generation tasks, supporting a wide range of LLMs using haystack, and integrating with HuggingFace DATASETS for easy evaluation.

Prompt-based learning (Liu et al., 2021; Gao et al., 2021; Schick and Schütze, 2021a; Le Scao and Rush, 2021) is another line of research that has proven useful in improving downstream tasks in zero- and few-shot settings by leveraging LLMs' pre-training objectives (Brown et al., 2020; Ouyang et al., 2022; Zhang et al., 2022; Scao et al., 2023; Touvron et al., 2023). However, the availability of training data in low-resource scenarios is still crucial (Perez et al., 2021; Sahu et al., 2022). Therefore, our method also seeks to fill this gap by providing a comprehensive and easily reproducible dataset generation toolkit.

5 Conclusion

We introduced FABRICATOR, a user-friendly library for dataset generation utilizing LLMs. With FABRICATOR, researchers access a highly customizable interface that enables efficient research on zero-shot and few-shot learning via dataset generation. Further, we implemented various baselines using generated datasets to illustrate potential applications of our repository and plan to support further downstream tasks in the future. We believe that FABRICATOR will be a valuable tool for the NLP community, facilitating advancements in dataset generation and fostering research in various natural language processing domains.

Limitations

While our paper aims to address dataset creation for a wide range of downstream tasks, it is important to acknowledge certain limitations in our study. Firstly, during our repository's evaluation phase, we could only test and assess a subset of tasks due to resource and time constraints. Our evaluation may only cover a portion of the tasks researchers and practitioners commonly encounter in their work. Future work must expand the evaluation to include a broader range of tasks to provide a more comprehensive understanding of the repository's effectiveness.

Additionally, despite our best efforts in designing the repository layout to be versatile and adaptable, there might be specific tasks or domains where our repository's structure or features may not be directly applicable. We acknowledge that the landscape of downstream tasks is diverse and constantly evolving, which may require tailored approaches or extensions to our existing framework. Further, we aim to include existing research targeting high-quality dataset generation (e.g., Ye et al. (2022b)) and conduct our own research on quality and diversity metrics to steer the generation process. We encourage open-source contributions and active engagement from the community to address these limitations. By involving a more comprehensive range of perspectives and expertise, we aim to consistently improve the repository and enhance its suitability for various task requirements.

Furthermore, while we have endeavored to provide thorough documentation and guidelines within the repository, there is always a possibility of overlooked issues or unforeseen challenges that may arise during dataset creation.

Ethics Statement

While large language models have shown remarkable advancements in natural language understanding and generation, their capabilities also raise important ethical considerations. One prominent concern is the potential for hallucination, where the models may generate false or misleading information. This aspect can have serious implications, especially when datasets are created for critical domains such as medicine, law, or journalism. It is crucial to exercise caution and verify the accuracy and reliability of outputs generated by our repository, particularly when making decisions that have real-world consequences.

Another ethical concern is the presence of biases in language models, which can perpetuate and amplify societal prejudices and inequalities. These biases can arise from biased training data (Haller et al., 2023) or biased patterns in human-generated text that the models learn from. Since our repository is in an early stage, we emphasize to carefully inspect created datasets to identify and rectify biases that may be present.

To ensure a responsible dataset creation process, it is essential to engage in thorough data validation, including identifying and addressing potential biases, checking data sources for reliability and credibility, and involving diverse perspectives in dataset collection and annotation processes. Moreover, continuous monitoring and auditing of the models' outputs and performance can help identify and rectify any ethical concerns arising during deployment.

Acknowledgements

We thank all reviewers for their valuable comments. Jonas Golde is supported by the German Federal Ministry of Economic Affairs and Climate Action (BMWK) as part of the project ENA (KK5148001LB0). Alan Akbik and Patrick Haller are supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Emmy Noether grant "Eidetic Representations of Natural Language" (project number 448414230). Alan Akbik is furthermore supported under Germany's Excellence Strategy "Science of Intelligence" (EXC 2002/1, project number 390523135). Felix Hamborg is supported by the WIN program of the Heidelberg Academy of Sciences and Humanities, financed by the Ministry of Science, Research and Arts of the State of Baden-Wurttemberg, Germany.

References

Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. 2022. Inpars: Unsupervised dataset generation for information retrieval. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2387–2392, New York, NY, USA. Association for Computing Machinery.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages

632–642, Lisbon, Portugal. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. 2023. An empirical survey of data augmentation for limited data learning in NLP. *Transactions of the Association for Computational Linguistics*, 11:191–211.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Haitao Zheng, and Maosong Sun. 2022. OpenPrompt: An open-source framework for prompt-learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 105–113, Dublin, Ireland. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3816–3830, Online. Association for Computational Linguistics.

Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. ChatGPT outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences*, 120(30).

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks are all you need.

Patrick Haller, Ansar Aynetdinov, and Alan Akbik. 2023. Opiniongpt: Modelling explicit biases in instruction-tuned llms.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

Teven Le Scao and Alexander Rush. 2021. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2627–2636, Online. Association for Computational Linguistics.

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 Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. Gpt understands, too.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. In *Advances in Neural Information Processing Systems*, volume 35, pages 462–477. Curran Associates, Inc.

Yu Meng, Martin Michalski, Jiaxin Huang, Yu Zhang, Tarek Abdelzaher, and Jiawei Han. 2023. Tuning language models as training data generators for augmentation-enhanced few-shot learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 24457–24477. PMLR.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

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

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True few-shot learning with language models. In *Advances in Neural Information Processing Systems*, volume 34, pages 11054–11070. Curran Associates, Inc.

Malte Pietsch, Timo Möller, Bogdan Kostic, Julian Risch, Massimiliano Pippi, Mayank Jobanputra, Sara Zanzottera, Silvano Cerza, Vladimir Blagojevic, Thomas Stadelmann, Tanay Soni, and Sebastian Lee. 2019. Haystack: the end-to-end NLP framework for pragmatic builders.

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.

Gaurav Sahu, Pau Rodriguez, Issam Laradji, Parmida Atighehchian, David Vazquez, and Dzmitry Bahdanau. 2022. Data augmentation for intent classification with off-the-shelf large language models. In *Proceedings of the 4th Workshop on NLP for Conversational AI*, pages 47–57, Dublin, Ireland. Association for Computational Linguistics.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, and Matthias Gallé et al. 2023. Bloom: A 176b-parameter open-access multilingual language model.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021b. Generating datasets with pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6943–6951, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. 2022a. ZeroGen: Efficient zero-shot learning via dataset generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11653–11669, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jiacheng Ye, Jiahui Gao, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2022b. ProGen: Progressive zero-shot dataset generation via in-context

feedback. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3671–3683, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models.

A Appendix

A.1 Screencast

A screencast about the FABRICATOR framework can be found on Vimeo.

A.2 Hyperparameters for Experiments

We used AdamW (Loshchilov and Hutter, 2019) as our optimizer with a batch size of 16. Further, we used a linear warm-up for 10% of the optimization steps. We fine-tune `roberta-base` for question answering with a learning rate of $1e^{-5}$ for two epochs without early stopping. For the `bert-base-uncased` PLM, we fine-tune using a learning rate of $2e^{-5}$ for either 5 (if training data has more than 1000 examples), 10 (if training dataset has at least 500 but less than 1001 examples) or 20 epochs (if training data is less than 501 examples). Further, across all experiments, we use 10% of the data as a validation split for model selection.

A.3 Generate Label-Conditioned Training Data

This experiment used label-conditioned generation to create new data for the TREC dataset containing six classes. To achieve this, we sampled a small few-shot dataset from the existing training split, consisting of 8 examples per class. During generation, for each label y, we included three uniformly sampled few-shot examples associated with that label. We generated 10k data pairs $(\mathbf{x}^g, y)$ and used them for fine-tuning. It is important to note that the gold-labeled dataset contains only around 3k examples. Thus the column "all" refers either to the 10k examples generated with GPT or to the ~3k gold-labeled examples. The experimental setup is identical to Section 3.

The results are depicted in Table 3. We observe significant performance drops compared to the re-annotation experiments for TREC from Section 3.1. For instance, using 10k generated examples achieves a performance level similar to using 50 human-annotated examples (compare to Table 1). However, we note that we performed no prompt optimization techniques or hyperparameter searches in all experiments. Additionally, we generated a uniform distribution of classes, while the gold-labeled dataset is skewed towards certain categories. It is worth mentioning that this class distribution information may not be available in real-world few-shot settings.

A.4 Impact of Few-Shot Examples on Label-Conditioned Generation

In this experiment, we generated 500 label-conditioned data pairs for the TREC dataset, following the approach described in Section 3.2. We conducted a sweeping analysis over two factors: the total number of few-shot examples per class and the number of few-shot examples included in the actual prompt.

The results are depicted in Table 4. Our findings show that including even a small number of few-shot examples (< 4) yields better results compared to generating without any few-shot examples. Moreover, when we used at least four examples per class, we observed significant improvements in the generation results, from 30.2 to 54.8 in accuracy ($\uparrow$ 24.6 pp. in accuracy). Additionally, using more examples in a distinct prompt slightly improved the model performance. We encountered one outlier when using 16 examples per class and including five examples in the prompt for generation, which resulted in lower performance than sampling from 8 few-shot examples per prompt. It is important to note that during this experiment, we did not adjust any hyper-parameters of the LLM for generation, such as temperature or top-k sampling.

A.5 Instruction-tuning open-source models

In this experiment, we compare the annotation performance of OpenAI's GPT-3.5 with an instruction-tuned open-source LLaMA model. To conduct this evaluation, we choose the token classification task on the CoNLL-03 dataset (Tjong Kim Sang and De Meulder, 2003), which generates one label for each token in the input, making it a structured task.

The results are shown in Table 5. We observe that using the dataset as-is results in often unusable annotation outputs, primarily due to imprecise formatting. To address this, we convert the token-level

Dataset	Data	# Training examples			
		50	500	1000	all
TREC-6	Gold	42.7 ± 9.6	93.8 ± 0.3	95.1 ± 0.6	97.1 ± 0.3
	Generated	27.5 ± 11.0	56.2 ± 3.3	57.9 ± 1.6	62.6 ± 3.4

Table 3: Results on TREC-6 with generated questions by GPT-3.5 using 3 few-shot examples (uniformly sampled from 8 possible few-shot examples per class). We observe that the generation performance is worse compared to an equally sized human-annotated dataset. However, the performance increases with the number of examples generated.

Dataset	# few-shot examples per class	# examples per class used in prompt				
		0	2	3	4	5
TREC-6	0	30.2 ± 0.6	-	-	-	-
	2	-	43.0 ± 3.7	-	-	-
	4	-	56.0 ± 0.5	56.3 ± 2.4	58.3 ± 2.2	-
	8	-	52.8 ± 1.5	58.8 ± 1.0	58.2 ± 1.0	64.0 ± 2.0
	16	-	58.3 ± 0.8	59.8 ± 2.5	58.7 ± 1.1	54.8 ± 1.5

Table 4: Results on 500 generated TREC-6 examples with different sizes of few-shot examples and number of few-shot examples included in the prompt. We observe that more few-shot examples result in better performance on the gold annotated test split.

Model	Acc.	(micro) F1
LLaMAv2 + Instr. Tuning	92.4	60.0
GPT-3.5*	88.4	52.5

Table 5: Comparison of instruction-tuned LLaMA models with 3-shot GPT-3.5 based on the training split of CoNLL-03. We report accuracy and span-level F1 score the annotation on the validation split. *: We convert tag sequences to spans in order to prompt the LLM with strings rather than sequence. However, 38% of the validation split annotations have different lengths after tokenization which have been filtered out for a fair comparison.

labels into spans and prompt the LLM to extract all named entities for the relevant categories. We then transform the found entities into token-level tags by searching for the annotations as substrings of the input text. We compare the performance of this approach with a instruction-tuned LLaMA model on the entire training split of CoNLL-03 by letting both LLMs annotate the validation set.

Unlike the previous evaluation, we did not train and evaluate a smaller PLM on the gold-labeled test set. Instead, we assess the performance between the gold-annotated validation split and the annotations made by the LLM. Our findings indicate that the annotation quality of instruction-tuned LLMs can significantly improve over OpenAI's GPT, as evident from the higher F1 score. This finding suggests that instruction-tuned models for dataset generation have the potential to facilitate the generation process for complex downstream tasks in future research endeavors.