LLMeBench: A Flexible Framework for Accelerating LLMs Benchmarking

Fahim Dalvi, Maram Hasanain, Sabri Boughorbel, Basel Mousi, Samir Abdaljalil, Nizi Nazar, Ahmed Abdelali[*], Shammur Absar Chowdhury, Hamdy Mubarak, Ahmed Ali, Majd Hawasly, Nadir Durrani, Firoj Alam
Qatar Computing Research Institute, HBKU, Qatar
{faimaduddin,fialam}@hbku.edu.qa

Abstract

The recent development and success of Large Language Models (LLMs) necessitate an evaluation of their performance across diverse NLP tasks in different languages. Although several frameworks have been developed and made publicly available, their customization capabilities for specific tasks and datasets are often complex for different users. In this study, we introduce the LLMeBench[1] framework, which can be seamlessly customized to evaluate LLMs for any NLP task, *regardless of language*. The framework features generic dataset loaders, several model providers, and pre-implements most standard evaluation metrics. It supports in-context learning with zero- and few-shot settings. A specific dataset and task can be evaluated for a given LLM in less than 20 lines of code while allowing full flexibility to extend the framework for custom datasets, models, or tasks. The framework has been tested on 31 unique NLP tasks using 53 publicly available datasets within 90 experimental setups, involving approximately 296K data points. We open-sourced LLMeBench for the community[2] and a video demonstrating the framework is available online.[3]

1 Introduction

The rapid advancement of sophisticated large language models (LLMs), supported by in-context learning (ICL) (Dong et al., 2023), has gained unprecedented popularity among both the research and development communities. The emergence of such large models facilitated diverse applications (Mousi et al., 2023). Given their success, a systematic evaluation and comparison against state-of-the-art is important to accurately gauge the potential of LLMs. A comprehensive evaluation allows understanding of the strengths and weaknesses of these models; guides us towards better human-LLMs interactions through prompting; and facilitates their broader applicability in different scenarios, especially in domains where safety and security are paramount concerns (e.g., healthcare, financial institutes) (Chang et al., 2023; Zhao et al., 2023; Zhu et al., 2023).

Numerous initiatives were launched to comprehensively assess the performance of LLMs on standard NLP tasks. The HELM project (Liang et al., 2022) conducted a thorough evaluation of LLMs for English, spanning various metrics and scenarios. Additionally, the BIG-Bench initiative (Srivastava et al., 2023) introduced an extensive evaluation of 214 tasks, even encompassing languages with limited resources. Notably, evaluations have been carried out on models like GPT2.5 (Radford et al., 2019), ChatGPT (OpenAI, 2023), and BLOOM (Scao et al., 2022) within multitask, multilingual, and multimodal settings. These evaluations were further extended to low-resource languages (Bang et al., 2023; Ahuja et al., 2023; Hendy et al., 2023; Khondaker et al., 2023).

Evaluating LLMs across diverse tasks often entails challenges related to costs, effort, and time

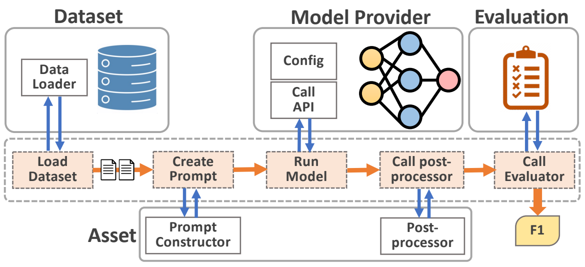

Figure 1: The architecture of the LLMeBench framework. The dotted boxes represent the core implemented modules of the architecture. Customization for new tasks, datasets, and models can be done on `Dataset`, `Model Provider`, `Evaluation`, and `Asset` modules.

[*]The contribution was made while the author was at the Qatar Computing Research Institute.

[1]**LLM e**ffectiveness **Bench**marking. Can be pronounced as “lemme bench”.

[2]https://github.com/qcri/LLMeBench/

[3]https://youtu.be/9cC2m_abk3A

due to complexities like handling API calls, task integration, dataset inclusion, evaluation measures, and potentially hosting datasets on public platforms (e.g., Hugging Face (HF)). To overcome these limitations, in this study, we introduce "LLMeBench", which facilitates a comprehensive evaluation of these LLMs through a seamless and flexible implementation. The proposed framework, as depicted in Figure 1, empowers users to assess various LLMs while simplifying the integration of custom tasks, datasets, and evaluation metrics.

A few evaluation frameworks have emerged to facilitate extensive benchmarking of LLMs. Among these are OpenAI evals,[4] LM Harness (Gao et al., 2021), and OpenICL (Wu et al., 2023). Each framework offers functionalities tailored to specific requirements. For instance, OpenICL focuses on few-shot learning techniques. Our contribution, LLMeBench, stands out by emphasizing a user-friendly, plug-and-play design, that can seamlessly integrate into existing experimental workflows, setting it apart from other alternatives. LLMeBench's uniqueness lies in the following features:

- Supports several generic data loaders (e.g., HF datasets), pre-implements several model providers (such as OpenAI and HF inference APIs for remote execution, and FastChat (Zheng et al., 2023) and Petals (Borzunov et al., 2023) for local deployments), and supports all standard tasks and evaluations, such as classification, regression, etc. Evaluating a new task/dataset/model can be done in as few as 20 lines.
- Allows the user to create their own data loader, connecting to their local server, ensuring data privacy and security.
- Provides users with the flexibility to design diverse tasks, allowing customization of data input/output formats and evaluation criteria.
- Supports zero- and few-shot learning paradigms with ∼300 zero-/few-shot prompts serving as a valuable community resource.
- Enables automatic selection of few-shot examples from a user-defined train/dev set using a maximal marginal relevance-based approach.
- Implements an efficient caching mechanism to prevent repeated API calls, resulting in cost savings and resolution of time-out issues.
- Offers extensive logging and caching capabilities, allowing iterative model outputs post-processing.
- Provides an auto-download mechanism for public datasets, accelerating experimentation.
- Includes 31 tasks recipes featuring different model providers. Rigorously tested with 53 datasets associated with 12 languages.

[4]https://github.com/openai/evals

```
class ModelBase(object):
    @abstractmethod
    def prompt(self, **kwargs):
        ''' Call to model API '''
        pass
    @abstractmethod
    def summarize_response(self, response):
        '''Extract response from model output'''
        pass
```

Listing 1: Abstract class for implementing a new `Model`.

Furthermore, LLMeBench is an open-source, user-friendly, and adaptable comprehensive benchmarking framework for LLMs. It empowers both experts and non-experts to assess conventional and unique NLP tasks, enhancing comprehension of the models' capabilities and their applicability across standard and novel tasks.

2 LLMeBench

In Figure 1, we provide the architecture of the LLMeBench framework. To ease the burden on users in implementing common elements across experimental setups, which are not specific to a task, the architecture was designed to support a uniform format for both input and intermediate outputs. This was achieved by employing a pipeline that utilizes key-value dictionaries to seamlessly pass data. The framework incorporates *four* fundamental modules, discussed below. The process starts with a `Dataset`, where each input sample S_i is routed to the `Asset` module. Within this module, a prompt is created and then passed to the `Model Provider` for processing. The model's response is then funneled back to the `Asset` module for post-processing. As the processing of all input samples and the generation of corresponding responses conclude, the `Evaluation` module takes on the task of computing evaluation metrics. The whole process of intercommunication is overseen by a `Benchmark Driver`. Throughout these processes, the inputs, processed data, and the intermediate outputs are cached for re-use and quick experimentation.

2.1 `Model Provider` module

A `Model Provider` abstracts away all model-specific communication and aims to set the defaults

```python
class DatasetBase(ABC):
    @abstractmethod
    def metadata(self):
        ''' Returns metadata for the dataset. '''
        pass
    @abstractmethod
    def get_data_sample(self):
        ''' Returns a single dictionary, with
            at least the following keys:
            'input': <input-instance>
            'label': <label> '''
        pass
    @abstractmethod
    def load_data(self, data_path):
        ''' Returns a list of dictionaries, with
            at least the following keys:
            'input': <input-instance>
            'label': <label> '''
        pass
```

Listing 2: Abstract class for implementing a new `Dataset`.

```python
class TaskBase(ABC):
    @abstractmethod
    def evaluate(self, true_labels,
                 predicted_labels):
        pass
```

Listing 3: Abstract class for implementing a new `Evaluation`.

for maximum reproducibility (for instance assign temperature value to zero by default). The framework currently supports OpenAI's API, the HF Inference API, as well as FastChat and Petals for local deployments. Defining a new LLM model is straightforward by extending the `ModelBase` class. The initial process entails configuring essential parameters for the model setup, including factors like temperature, top_p, etc. Furthermore, it requires the implementation of two abstract methods (shown in Listing 1): `prompt` – manages the invocation of the model API based on the input prompt; and `summarize_response` – extracts a summary of the response from raw model output.

2.2 `Dataset` module

Similar to a `Model provider`, a `Dataset` implementation aims to abstract dataset-specific code, such as loading, pre-processing, and formatting of samples. The framework comes with four generic data loaders, including Hugging Face, CSV, and JSON datasets. A custom dataset can be easily implemented by extending the `DatasetBase` class. When defining a new dataset, the user is required to implement at least three methods (depicted in Listing 2). The first, `metadata`, is designed to provide comprehensive metadata such as a citation or

```python
def config():
    return {
       'dataset':ExampleDataset,'dataset_args':{},
       'task':ExampleTask,'task_args':{},
       'model':OpenAIModel,'model_args':{},
       'general_args':{}}
def prompt(input_sample):
    ''' Construct and return the prompt following
        the model's corrsponding template'''
def post_process(response):
    ''' Apply custom post-processing on response
        and return extracted model predicition'''
```

Listing 4: Methods to implement in the `Asset` module.

reference to the source of the dataset, its download link, and the languages it covers. The second function to be implemented in this module is `load_data`, which should define a data loader capable of returning a list S comprising the samples from the dataset, given the user-specified dataset path. Lastly, `get_data_sample` should be defined to return a Python dictionary representing a single sample S_i extracted from the dataset.

2.3 `Evaluation` module

The `Evaluation` module aims to compute metrics and consolidate the results for a task. The framework comes with built-in support for popular task types such as Classification and Regression and is easily extendible to any custom metric by inheriting from the `TaskBase` class. A custom implementation can define an `evaluate` function for a task with specific evaluation code and metrics (see Listing 3). The function is passed two lists: the predicted labels, and true or gold labels. Its primary objective is to yield a user-defined Python dictionary comprising key-value pairs representing the outcomes of the evaluation (e.g., {"`Accuracy`": `accuracy value`}).

2.4 `Benchmarking Asset module`

The `Asset` module represents a benchmarking experiment, utilizing all the modules defined in LLMeBench as can be seen in code snippet Listing 4. Within this module, the user should provide full configuration for the experiment, which includes specifying the `Dataset`, `Model`, and `Evaluation` modules. The module also enables passing model and dataset parameters.

The `Asset` module must also implement the `prompt` function, which constructs the actual prompt to pass to the `Model`, based on the input sample. For scenarios involving few-shot learning, the `Asset` module is provided with k examples to use in prompt construction. These examples are chosen by the framework from a training dataset specified by the user, where k is a parameter controlled by the user.

Finally, the `post_process` function is required to be implemented to post-process response from the model. This step is crucial because the output produced by the `Model` is tailored to the particular model and the prompt used, leading to potential variations across benchmarking experiments.

2.5 Interaction

Once the aforementioned modules are implemented, running a benchmarking experiment becomes a straightforward task, accomplished through a single command that provides access to various adjustable parameters. The package automatically identifies the `Asset` to run based on wildcard search using the provided asset name. Additionally, the package determines whether to activate the few-shot setup when the number of shots is specified as a parameter (`--n_shots` <k>). Furthermore, the package supports swift testing of the benchmarking asset by executing it on a small number of n (`--limit` <n>) samples, which limits the run to the first n samples from the dataset. An example of the command is provided below.

```
$ python -m llmebench --filter '*AssetName*'
--n_shots k --limit n --ignore_cache
<benchmark-dir> <results-dir>
```

3 Features

LLMeBench features a generic framework that serves a broad range of tasks and models in different learning settings (zero-/few-shot) to evaluate model performance. It enables scalable and rigorous evaluation across diverse tasks and languages while offering simplicity of implementation and flexibility in customization.

3.1 Modularity

The LLMeBench framework, as shown in Figure 1, follows loosely-coupled design principles, effectively separating the data loader, models, and evaluation components. These components interact through a `Benchmark Driver`, ensuring a modular and flexible architecture.

3.2 Generality

The framework is designed to offer generality, with effortless customization of tasks, data, and models. The framework comes with several generic data loaders such as Hugging Face datasets. Additionally, since users have the ability to create their own data loaders, the framework can support any standard data format. At the time of writing this paper, we have conducted tests with different formats including *TSV*, *CSV*, *JSON*, and *JSONL*.

In terms of tasks, the framework demonstrates the capability to handle a diverse array of token and sequence classification tasks. Figure 2 displays the implemented task types, with built-in support for all standard generic tasks. Additionally, any new custom task can be seamlessly incorporated.

The framework also accommodates various types of models, encompassing both open and closed models, each with its own API-based options. For closed models, users can acquire API keys and endpoints from hosting providers. Meanwhile, open models can be hosted on in-house infrastructure or any accessible hosting service through APIs. We evaluated such an integration using BLOOMZ 176B 8bit version and Jais-13b (32 bit) (Sengupta et al., 2023), hosting it within our in-house infrastructure using Petals (Borzunov et al., 2023) and FastChat. Overall, the framework offers a high level of generality and can be readily applied and adapted to a wide range of use cases and research scenarios.

3.3 Prompts

LLMeBench is designed to support both zero- and few-shot learning setups. The instructions in prompts can be written in any language of interest.

Zero-shot prompts provide natural language instructions describing the task and expected output.

Few-shot prompts embed a small number of examples to the natural language instructions for the particular task. The framework utilizes user-defined, task-specific training set to automatically select few-shot examples. Various strategies exist for examples selection. Among these, we have implemented maximal marginal relevance-based (MMR) (Carbonell and Goldstein, 1998) selection, which has demonstrated success in previous work by Ye et al. (2023). The approach computes the similarity between a test example and the example pool (e.g., training dataset) and selects k examples (shots) that are both relevant and diverse. We apply the MMR technique on top of embeddings obtained from the multilingual sentence-transformers (Reimers and Gurevych, 2019). How-

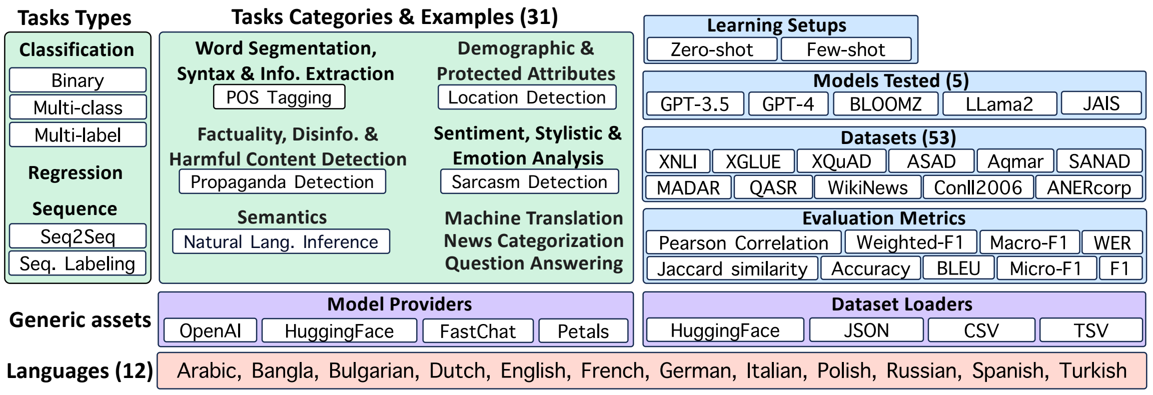

Figure 2: Summary and examples of the 53 datasets, 31 tasks, 4 model providers, 5 tested models and metrics currently implemented and validated in LLMeBench.

ever, users also have the flexibility to utilize any custom embedding model.

We have designed a highly efficient process to extract embeddings and compute few-shot examples for each test sample. Specifically, it pre-selects few-shot examples for each test sample during the initial loading stage. This design effectively eliminates the need to apply and compute the MMR score when making API calls, thus enhancing the system's overall efficiency.

The LLMeBench framework includes ∼300 designed prompts for zero-shot and few-shot setups that have been validated across a variety of NLP tasks and datasets (see Section 4). This collection serves as a strong starting point for the community and expedite prompt engineering research.

3.4 Caching

One of the significant challenges when accessing APIs is managing timeout issues. The necessity to rerun experiments involving API calls not only requires additional effort but also increases costs. To address this problem, we have developed a caching mechanism, allowing users to bypass making API calls for samples that have already been successfully processed. Specifically, we save all intermediate outputs when processing a data sample, including the generated prompt, the raw model response and the post-processed output. On a re-run, samples that have model responses in the cache do not actually access the API, but rather load the cached response. Furthermore, this caching mechanism plays a vital role in enhancing the post-processing of the models' output, as it can be performed repeatedly without having to call the API. This feature is important, given that different models yield various types of outputs, requiring improved post-processing to align the output accurately with the reference label. To further counter expected timeout and rate limitation issues with APIs, the framework also applies a user-configurable *wait-and-retry* mechanism. This mechanism retries API calls in case of failure, maximizing the chance of receiving a successful response.

3.5 Dataset Auto-Download

The framework comes with support for automatic downloading and caching of publicly available datasets, taking care of extracting and linking them correctly for any `Asset` that requires it. This allows a new user to quickly begin experimenting with and evaluating an existing dataset without the need to manually acquire it first.

3.6 Task Diversity

The framework currently supports and includes a diverse set of tasks, covering a broad spectrum that ranges from word-level tasks to those involving single sentences, sentence pairs, question-answer pairs, and more. The range of tasks covers various NLP research tracks, as can be seen in Figure 2.

3.7 Language-agnostic Framework

The LLMeBench framework is language-agnostic. As of now, tasks for 12 languages have been incorporated, a number that will continuously grow as a result of our ongoing efforts and, ideally, with the support of the community.

3.8 Open-source Framework

We made *LLMeBench* accessible to the community by releasing it as open-source. This will also enable the continued growth and development of the framework within the community.

3.9 Deploying Local Model

For deploying local models, we interface LLMeBench with FastChat framework (Zheng et al., 2023). The latter is an open-source project [5] for serving LLMs with a fast-growing user community. Custom chat templates of popular and new LLMs are rapidly added to the framework. This allows a proper use of instruct-tuned LLMs in inference mode, unlike other popular benchmarking frameworks such Eval-Harness (Gao et al., 2021), which do not use chat templates. This can lead to a mismatch in the expected inputs of instruction tuned models, and put at disadvantage several LLMs, especially the ones with small sizes. Our approach for local deployment solves this issue by relying on FastChat. Huggingface models can be easily deployed in three steps after installing FastChat python package: 1) Running a model controller which plays a role of interfacing between API and model calls. 2) Running a model worker for each loaded model which manages the model in GPU and executes the prompts and returns to responses to the model worker. It is possible to load model worker with vLLM [6] enabling prompt batching for an efficient and fast inference (Kwon et al., 2023). 3) Running an API server which provides a compatible interface with to OpenAI API. The local address and port of the API are set in LLMeBench via `FASTCHAT_*` environment variables.

4 Evaluation of LLMeBench

The framework has already been used across a variety of Arabic NLP tasks and datasets (Abdelali et al., 2024). This involved extensive experimentation using zero- and few-shot learning with state-of-the-art large language models, including GPT-3.5-Turbo, GPT-4, and the 8-bit version of the BLOOMZ 176B model. In Figure 2, we provide a summary of the tasks, datasets, and models that have been implemented and evaluated. Given that our assessment of the framework was based on the current state-of-the-art NLP tasks and datasets, we implemented task- and dataset-specific metrics reported in the literature. Overall, it has been used to evaluate 31 NLP tasks, which were categorized based on ACL tracks, 53 datasets, and different model providers within 2 learning setups. All the task recipes are available within the framework.

[5]https://github.com/lm-sys/FastChat

[6]https://github.com/vllm-project/vllm/

5 Related Work

Efforts to assess the performance of LLMs on standard NLP tasks have been underway since the launch of ChatGPT. Notable studies, such as those by Bubeck et al. (2023), Bang et al. (2023), Ahuja et al. (2023), and Hendy et al. (2023), have conducted large-scale experiments considering multilinguality, multimodality, low-resource languages, and a wide range of datasets and tasks.

Such large-scale evaluations require off-the-shelf and easy-to-use solutions to measure the performances of LLMs for a variety of NLP-related tasks. To evaluate OpenAI's models, the company developed the EVALs[7] package, which requires the dataset to be prepared in JSON format using a predefined template. Asai et al. (2023) developed an evaluation framework as a part of their cross-lingual benchmarking effort. This comprehensive framework includes 15 distinct tasks set in a few-shot learning environment across 54 languages. However, this evaluation framework was not publicly available at the time of writing this paper. OpenICL (Wu et al., 2023) is another framework designed specifically for zero-shot or few-shot learning setups. It incorporates various strategies, such as random selection, heuristic methods (including BM25 (Robertson et al., 2009), TopK (Liu et al., 2022), and VoteK (Hongjin et al., 2022)), and a model-based approach, to select few-shot examples. The OpenICL framework is implemented under the assumption that users will utilize HF datasets to load and evaluate models. A prompt is an important part that serves as a bridge between humans and LLMs. To explore the research in this direction Zhu et al. (2023) developed the PromptBench framework for prompt engineering. Eleuther-AI developed LM evaluation Harness (Gao et al., 2021), which includes the implementation of 200 tasks and supports multiple models from the HF hub.

	Customization			ICL (shot)	
Eval Package	**Dataset**	**Task**	**Models**	**Zero**	**Few**
OpenAI evals[8]	Fixed	✓	Any	✓	✓
LM Harness (Gao et al., 2021)	HF	✓	HF	✓	✓
OpenICL (Wu et al., 2023)	HF	✓	HF, OpenAI	✓	✓
LLMeBench (Ours)	Custom	✓	Any	✓	✓

Table 1: LLMs evaluation frameworks. HF: Hugging Face

Compared to the aforementioned frameworks (summarized in Table 1), the LLMeBench frame-

[7]https://github.com/openai/evals

work offers customization through custom dataset loaders, tasks, and models. It also supports both zero- and few-shot prompting. The caching mechanism provided by the LLMeBench framework is a rarity among its counterparts, yet it is crucial for time and cost savings, as well as facilitating efficient post-processing enhancements to model outputs without additional expenses.

6 Conclusions and Future Work

In this paper, we introduce LLMeBench, an open-source framework designed to facilitate the LLM benchmarking process. LLMeBench accelerates evaluation of LLMs using pre-implemented generic datasets, tasks and model providers. In addition, it features a modular design that empowers users to integrate *(i)* new tasks, *(ii)* datasets, and *(iii)* APIs for models. The framework incorporates caching mechanisms that effectively reduce time, costs, and effort associated with task evaluations. Currently, it includes predefined recipes covering 31 standard NLP tasks such as classification, translation, question-answering, and semantic parsing. These recipes can be readily extended to encompass novel NLP tasks, datasets, and LLM models.

In future, we aim to further enhance the framework by integrating a broader array of tasks and languages. By embracing an open-source approach and encouraging active community participation, we anticipate its sustained growth through the incorporation of diverse tasks, enriched datasets, and innovative models. Additional enhancements under consideration encompass integrating cross-validation datasets, and incorporating models featuring varied configurations (such as distinct iterations of BLOOM models). Furthermore, we are actively developing more methods for few-shot selections. Currently, our framework assumes seamless model access via APIs. We are committed to enhancing accessibility by enabling users to effortlessly load and utilize both offline and online models for inference purposes.

Limitations

The LLMeBench is currently limited to API calls, whether they are local or remotely hosted. It also operates under the assumption that the entire dataset can fit into memory, which may not be feasible for very large collections. Implementing iterable loading could be a viable solution to this issue, and is a feature that might be considered for future development. Additionally, many datasets come with cross-validation splits, a functionality that the framework does not currently support.

Ethics Statement

Our framework incorporates publicly available datasets and relies on external models. These models may produce non-factual or potentially harmful content. Therefore, we encourage users to be aware of their interaction with the models.

Acknowledgments

The contributions of M. Hasanain were funded by the NPRP grant 14C-0916-210015, which is provided by the Qatar National Research Fund (a member of Qatar Foundation).

References

Ahmed Abdelali, Hamdy Mubarak, Shammur Absar Chowdhury, Maram Hasanain, Basel Mousi, Sabri Boughorbel, Samir Abdaljalil, Yassine El Kheir, Daniel Izham, Fahim Dalvi, Majd Hawasly, Nizi Nazar, Yousseif Elshahawy, Ahmed Ali, Nadir Durrani, Natasa Milic-Frayling, and Firoj Alam. 2024. LAraBench: Benchmarking Arabic AI with Large Language Models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Malta. Association for Computational Linguistics.

Kabir Ahuja, Harshita Diddee, Rishav Hada, Millicent Ochieng, Krithika Ramesh, Prachi Jain, Akshay Nambi, Tanuja Ganu, Sameer Segal, Mohamed Ahmed, Kalika Bali, and Sunayana Sitaram. 2023. MEGA: Multilingual evaluation of generative AI. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4232–4267, Singapore. Association for Computational Linguistics.

Akari Asai, Sneha Kudugunta, Xinyan Velocity Yu, Terra Blevins, Hila Gonen, Machel Reid, Yulia Tsvetkov, Sebastian Ruder, and Hannaneh Hajishirzi. 2023. BUFFET: Benchmarking large language models for few-shot cross-lingual transfer. *arXiv preprint arXiv:2305.14857*.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 675—-718, Indonesia. Association for Computational Linguistics.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Maksim Riabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. 2023. Petals: Collaborative inference and fine-tuning of large models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 558–568, Toronto, Canada. Association for Computational Linguistics.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with GPT-4. Technical report, Microsoft Research.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2023. A survey on evaluation of large language models. *arXiv preprint arXiv:2307.03109*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation. *Zenodo*.

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How good are GPT models at machine translation? a comprehensive evaluation. *arXiv preprint arXiv:2302.09210*.

SU Hongjin, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022. Selective annotation makes language models better few-shot learners. In *The Eleventh International Conference on Learning Representations*.

Md Tawkat Islam Khondaker, Abdul Waheed, El Moatez Billah Nagoudi, and Muhammad Abdul-Mageed. 2023. GPTAraEval: A comprehensive evaluation of ChatGPT on Arabic NLP. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 220–247, Singapore. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B 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.

Basel Mousi, Nadir Durrani, and Fahim Dalvi. 2023. Can LLMs facilitate interpretation of pre-trained language models? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3248–3268, Singapore. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. Technical report, OpenAI.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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 and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Neha Sengupta, Sunil Kumar Sahu, Bokang Jia, Satheesh Katipomu, Haonan Li, Fajri Koto, Osama Mohammed Afzal, Samta Kamboj, Onkar Pandit, Rahul Pal, et al. 2023. Jais and jais-chat: Arabic-centric foundation and instruction-tuned open generative large language models. *arXiv preprint arXiv:2308.16149*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Zhenyu Wu, Yaoxiang Wang, Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Jingjing Xu, and Yu Qiao. 2023. OpenICL: An open-source framework for in-context learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 489–498, Toronto, Canada. Association for Computational Linguistics.

Xi Ye, Srinivasan Iyer, Asli Celikyilmaz, Veselin Stoyanov, Greg Durrett, and Ramakanth Pasunuru. 2023. Complementary explanations for effective in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4469–4484, Toronto, Canada. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Neil Zhenqiang Gong, Yue Zhang, et al. 2023. PromptBench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv preprint arXiv:2306.04528*.