
Reproducing Guideline for Benchmarking LLMs via Uncertainty Quantification

Our implementation under the MIT license is available at both the supplementary materials enclosed and the GitHub repo <https://github.com/smartyfh/LLM-Uncertainty-Bench>.

1 Data preparation

In this work, we propose a new method that takes into account uncertainty quantification to benchmark LLMs. Our benchmarking relies on existing publicly available datasets, including

- **MMLU**: MMLU was released by Hendrycks et al. (2020) and is available at <https://github.com/hendrycks/test>.
- **CosmosQA**: CosmosQA was released by Huang et al. (2019) and is available at <https://github.com/wilbur0ne/cosmosqa>.
- **HellaSwag**: HellaSwag was released by Zellers et al. (2019) and is available at <https://github.com/rowanz/hellaswag>.
- **HaluDial**: HaluDial was built upon HaluEval, released by Li et al. (2023) and is available at <https://github.com/RUCAIBox/HaluEval>.
- **HaluSum**: HaluSum was built upon HaluEval as well.

The licenses of these datasets are provided in the following table.

Table 1: Licenses of datasets.

MMLU	CosmosQA	HellaSwag	HaluDial	HaluSum
MIT License	None	MIT License	MIT License	MIT License

All these datasets are in the form of multiple-choice question answering and there is only one correct option for each question. However, these raw datasets have different data formats and different number of data points. They also have different number of answer choices for each question. To facilitate comparisons among different tasks, we standardize these datasets in the json format and keep only **10,000** randomly sampled data points for each dataset. Moreover, we standardize the number of options for each question to 6 and the last two options are always "*I don't know*" and "*None of the above*". Our implementation of this data preprocessing is provided in the `data_preparation` folder. Considering that it is time-wasting to repeat this process, we have also provided the preprocessed datasets, which are available in the `data` folder.

2 Prompting LLMs to obtain predicted probabilities for options

The next step is to prompt a particular LLM to obtain the predicted probabilities for all options of each question. Specifically, instead of acquiring these probabilities from the LLM directly, we obtain the logits corresponding to each option letter from the `LM_Head` layer. Then, we convert these logits to probabilities by applying the softmax function. It is worth emphasizing that since we rely on the

output logits to calculate predicted probabilities, we just need to perform the forward pass once and there is no need to do any sampling. In other words, the obtained results are deterministic.

Considering that LLMs are sensitive to different prompts, we have proposed three prompting strategies: *base prompt*, *shared-instruction prompt*, and *task-specific instruction prompt*. We take the average results corresponding to these three prompting strategies to mitigate the influence of LLMs' sensitivities to prompts. The detailed prompts can be found in the `prompt.py` file.

Our implementation of prompting LLMs can be found in the `generate_logits.py` file. In particular, it can be run as follows:

```
python generate_logits.py \  
  --model={path to model directory} \  
  --data_path={path to data directory} \  
  --file={name of dataset} \  
  --prompt_method={base/shared/task} \  
  --output_dir={output directory} \  
  --few_shot={1 for few-shot and 0 for zero-shot}
```

In our experiments, we have adopted the few-shot setting (i.e. in-context learning) to help LLMs achieve better performance.

The above implementation is only applicable to pre-trained LLMs. Given that the instruction-finetuned (chat) version is more useful in practice, we also assess the uncertainty of instruction-finetuned LLMs. For this purpose, we provide a chat version of the `generate_logits.py` file, namely `generate_logits_chat.py` file. This file can be run as follows:

```
python generate_logits_chat.py \  
  --model={path to model directory} \  
  --data_path={path to data directory} \  
  --file={name of dataset} \  
  --prompt_method={base/shared/task} \  
  --output_dir={output directory} \  
  --few_shot={1 for few-shot and 0 for zero-shot}
```

3 Applying conformal prediction for uncertainty quantification

The last step is to quantify uncertainty based on the predicted probabilities. We choose to employ conformal prediction Angelopoulos et al. (2023) as the uncertainty quantification method. Conformal prediction is a **distribution-free** and **model-agnostic** approach to uncertainty quantification. It can transform any heuristic notion of uncertainty from any model into a statistically **rigorous** one. For multi-class classification tasks, conformal prediction outputs a prediction set of possible labels (answers) that encompasses the correct label with a user-specified error rate α and expresses uncertainty as the set size. Intuitively, a larger set size indicates higher uncertainty and vice versa.

Our implementation can be found in the `uncertainty_quantification_via_cp.py` file. Specifically, it can be run as follows:

```
python uncertainty_quantification_via_cp.py \  
  --model={model name} \  
  --raw_data_dir={path to data directory} \  
  --logits_data_dir={path to the directory where option logits are stored} \  
  --data_names={list of datasets to be evaluated} \  
  --cal_ratio={how much data to be used as the calibration data, e.g., 0.5} \  
  --alpha={error rate, e.g., 0.1}
```

It is noted that there is an argument `cal_ratio`, which denotes the portion of data used as calibration data. This is because conformal prediction splits the dataset into a calibration set and a test set. The reported results are derived from the test set.

References

- Angelopoulos, A. N., Bates, S., et al. (2023). Conformal prediction: A gentle introduction. *Foundations and Trends® in Machine Learning*, 16(4):494–591.
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. (2020). Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.
- Huang, L., Le Bras, R., Bhagavatula, C., and Choi, Y. (2019). Cosmos qa: Machine reading comprehension with contextual commonsense reasoning. 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 2391–2401.
- Li, J., Cheng, X., Zhao, W. X., Nie, J.-Y., and Wen, J.-R. (2023). Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6449–6464.
- Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.