LLMBlender++

Doddaka Venkata Sai
IIT Guwahati
234156021
d.venkatasai@iitg.ac.in

Vishal Narayan Deka
IIT Guwahati
234156031
vishal.deka@iitg.ac.in

Seetha Ramanjaneya Prasadarao Puligadda
IIT Guwahati
234156030
p.seetha@iitg.ac.in

ABSTRACT

LLMBlender++ is an attempt at a course project for Neural Networks for NLP (CS-563). Current open-source LLMs are trained on varied datasets and perform differently across different text generation domains. A committee of such LLMs can therefore be ensembled to attain a better output than when only a single LLM is used. Furthermore, the ensembling framework used for this work, namely LLMBlender, is stateless. A modification to this framework has been implemented, resulting in a stateful model: the LLMBlender++. LLMBlender++ gives a user the flexibility to deploy any number of constituent LLMs in the committee as he/she requires and can ask follow-up questions that return coherent outputs.

ACM Reference Format:
Doddaka Venkata Sai, Vishal Narayan Deka, and Seetha Ramanjaneya Prasadarao Puligadda. 2024. LLMBlender++. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 PROBLEM TO BE SOLVED

Currently, most of the state of the art and prominent LLMs out in the market are close-sourced and their architectures and training datasets are not readily transparent to the public. As a result, many researchers and enthusiasts have turned to/developed smaller local LLMs. These LLMs are fine-tuned and trained on custom-instruction datasets and hence, develop complementary strengths. With such LLMs possessing heterogeneous expertise in a vast array of fields, it can be argued that an ensemble of these LLMs can lead to consistently better outputs. Our work aims to leverage these strengths of multiple smaller LLMs to generate better and coherent text than would otherwise be possible when generating from a lone LLM. As their unique contributions are merged, biases, errors, and uncertainties in individual LLMs are alleviated.

2 IMPORTANCE OF THE PROBLEM

There's a number of smaller, open-source LLMs available for use. While these local LLMs might not offer better inference than LLMs like ChatGPT or Co-Pilot, they still have many advantages, a couple of them being:

- Fine-tuning and training for very specific use-cases.
- Avoiding the costs/dependency of running inference on third party providers, a sort of future proofing.
- Increased data security, avoiding leakages, data corruption and many more.

The motivations behind using local LLMs are hopefully evident. There's considerable work being carried out currently to improve inference of these local LLMs, as more clients explore ways to offload off 3rd party platforms into local deployments.
The solutions explored in this work can fall under LLM 'ensembling'. Beyond the traditional approach of training an LLM from scratch, because of their functional and structural differences, an alternative option is to combine existing LLMs into a new, more powerful one. This route was proposed in the Knowledge Fusion of Large Language Models[3] paper, which mainly dealt with the manipulation of token probability distribution matrices to further train one of the LLMs in an ensemble so that it can become a *FuseLLM*. Another way of ensembling proposed by Lu et al is the *Zooter*[2]. The Zooter is a reward-guided routing method distilling rewards on training queries to train a routing function, which can precisely distribute input queries to the LLM with expertise about it. In this form of ensembling, there's no fusion or merging of outputs happening, and is rather an algorithm that allows the committee to 'play to their strengths'. We have looked at a third way of ensembling, namely LLMBlending in our work.

3 INTUITION

3.1 LLMBlender

LLMBlender is an LLM ensembling framework developed by Jiang et al that is capable of generating consistently better performance when outputs of multiple LLMs are combined [1]. It consists of 2 modules, which compare outputs of N constituent LLMs before merging the best K among them, respectively. These modules are:

3.1.1 PairRanker. The PairRanker is responsible for ranking outputs of N LLMs in the ensemble/committee. PairRanker, as the name suggests, compares LLM outputs in pairs instead of calculating a score for each sequentially. This method of comparison allows for more nuanced discerning of individual LLM capabilities.
Of the N LLMs in the committee, *C(N,2)* combinations of pairs are taken. The input query x and the candidate texts *yi* and *yj* are then jointly encoded and sent as input to a cross-attention encoder RoBERTa in order to determine which among the two is better.

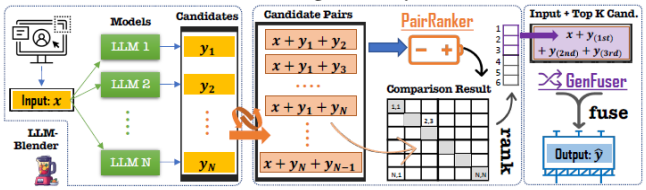
3.1.2 GenFuser. Merely selecting the best performing output via the PairRanker module might constrain the potential of generating better results than the candidates, as it is possible that different candidates might have distinct correct/coherent portions in their outputs. The solution proposed is therefore a Generative Fuser

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(GenFuser for short), which fuses the top K ranked candidates out of N, to generate an output that builds on the strengths of the LLMs while mitigating their weaknesses.

More technically, GENFUSER is a seq2seq approach for fusing a set of candidates given an input instruction to generate an enhanced output. The developers of LLMBlender concatenated the input and K candidates sequentially using separator tokens and fine-tuned a Flan-T5-XL instance to learn to generate y.

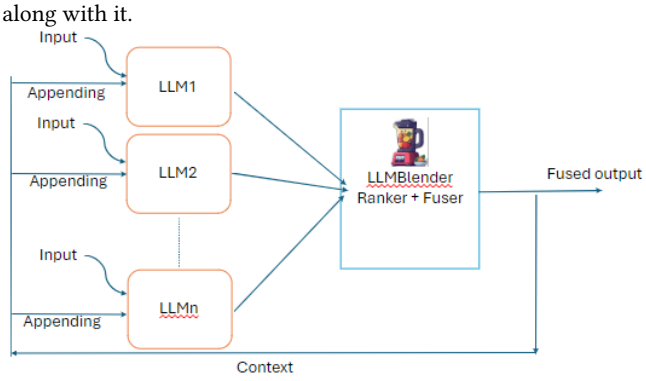


The architecture of LLMBlender

3.2 LLMBlender++

LLMBlender is a stateless framework, as its constituent LLMs are themselves stateless. The context window is by default limited to only that particular query sent into the framework and as a result, a conversational interaction cannot be had with LLMBlender without making some modifications. We have implemented a feedback scheme onto the LLMBlender framework, enabling constituent LLMs to become aware of the best (fused) output that was generated downstream and incorporate that information when answering the next follow-up query. This extension over LLMBlender has been termed as LLMBlender++.

In order to implement this feedback mechanism, we have resorted to a simple method very similar to the buffer conversational memory type supported in LangChain, which is passing raw conversation history concatenated with input as the next query. In other words, when the next follow-up query is sent, all past information is sent along with it.



The architecture of LLMBlender++

3.3 Methodology

We intended to gauge whether LLMBlender++ results in better outputs owing to conversational ability, and if so by how much, when compared to the original work of LLMBlender. The entire codebase was executed on a Google Colab environment (free version), making use of local LLMs running in an Ollama server and a python script to build the feedback architecture and to automate querying.

We have selected 4 local LLMs for the study, which are marked under Text Generation tag in HuggingFace These models are:

- Gemma - 2B
- Mistral - 7B
- Qwen - 4B
- Llama2 - 7B

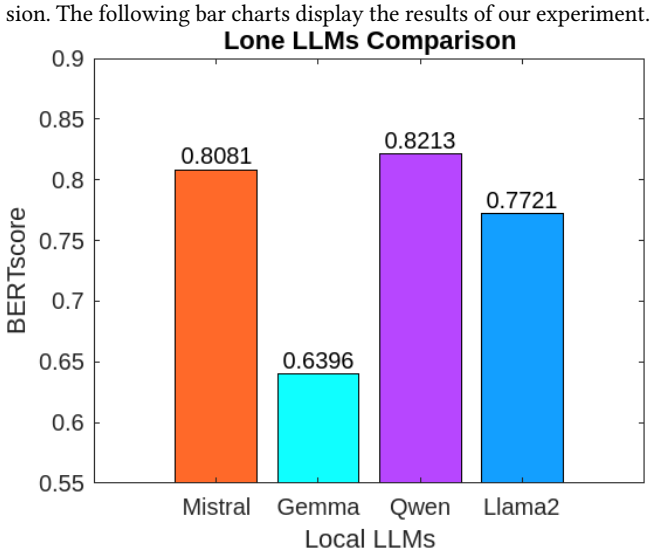
Since it can be difficult to say what's 'correct' or 'better' in a text generation task, we have picked the **conv_questions** dataset from HuggingFace for our study. This dataset contains sets of a leading question and a couple of follow-ups (10,000 sets in total), which were perfect for demonstrating statefulness achieved in LLMBlender via our modifications. Due to time and GPU constraints, we randomly sampled 10 question sets (40 questions in total) to test LLMBlender++. To demonstrate the efficacy of LLMBlender++, we carried out 3 experiments:

- query each LLM individually with questions from the dataset, with no feedback
- query LLMBlender on the dataset
- query LLMBlender with feedback architecture, i.e. LLMBlender++ on the dataset

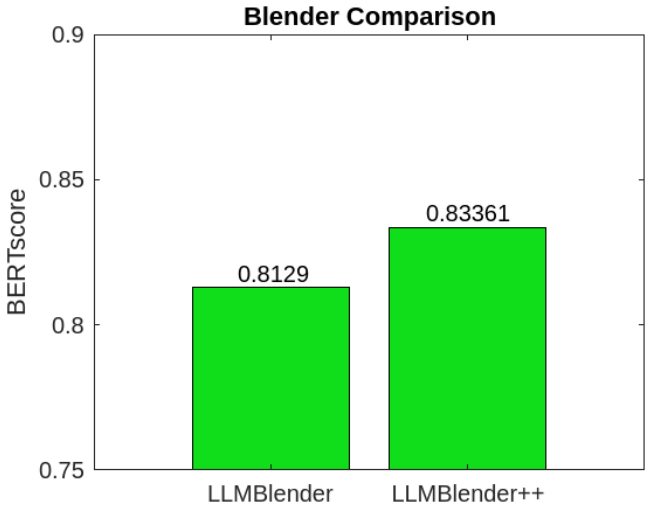
After collecting outputs of the above three experiments, we calculated the BERTscore as means of comparing the final outputs.

4 RESULTS

The results collected corroborate the intuition that LLMBlender consistently outperforms its constituent LLMs, and that LLMBlender++ gives better outputs, albeit slightly, compared to its stateless version. The following bar charts display the results of our experiment.



The BERTscore used here is a pretrained model, and is not fine-tuned on downstream tasks specific to this setting. With enough relevant verbosity in the output texts, BERTscores can be influenced positively and that possibly explains why these scores are relatively high even though there were a large number of 'confused' outputs seen due to absence of context feedback.



Moreover, while it may seem that there's barely an improvement as we move from LLMBlender to LLMBlender++, it is worth noting that highest BERTscore of individual question sets reached 0.92 in LLMBlender++ but only 0.85 in LLMBlender. Regardless, the trends seen do indicate an increase in quality of results.

All our code has been uploaded to a GitHub repo and can be found in the link: CS-563-LLMBlender_PlusPlus

5 LIMITATIONS

The limitations in our work are listed below:

- LLMBlender++ has to append fused output as context for the next query after every generation. This takes a significant amount of time and is one of the main factors limiting the size of our dataset during experimentation.
- The PairRanker module is essentially a RoBERTa model and is queried C(N,2) times for N candidates. This is a significant amount of extra computation required to make LLMBlender++ work.
- Since constituent LLM architectures can be very varied, the entire framework can be held up by the slowest performing LLM.
- The BERTscore used for comparing results is pre-trained and not fine-tuned for this specific experiment.
- As is the case with remembering context in the manner discussed in our work, LLMBlender is prone to forget previous information as input tokens increase. The context window lengths of constituent LLMs are not infinite.

REFERENCES

[1] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. arXiv:2306.02561 [cs.CL]

[2] Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. Routing to the Expert: Efficient Reward-guided Ensemble of Large Language Models. arXiv:2311.08692 [cs.CL]

[3] Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. 2024. Knowledge Fusion of Large Language Models. arXiv:2401.10491 [cs.CL]