LLM Evaluate: An Industry-Focused Evaluation Tool for Large Language Models

Harsh Saini, Md Tahmid Rahman Laskar, Chen Cheng, Elham Mohammadi, David Rossouw
Dialpad Inc.
{hsaini, tahmid.rahman, cchen, elham.mohammadi, davidr}@dialpad.com

Abstract

Large Language Models (LLMs) have demonstrated impressive capability to solve a wide range of tasks in recent years. This has inspired researchers and practitioners in the real-world industrial domain to build useful products via leveraging LLMs. However, extensive evaluations of LLMs, in terms of accuracy, memory management, and inference latency, while ensuring the reproducibility of the results are crucial before deploying LLM-based solutions for real-world usage. In addition, when evaluating LLMs on internal customer data, an on-premise evaluation system is necessary to protect customer privacy rather than sending customer data to third-party APIs for evaluation. In this paper, we demonstrate how we build an on-premise system for LLM evaluation to address the challenges in the evaluation of LLMs in real-world industrial settings. We demonstrate the complexities of consolidating various datasets, models, and inference-related artifacts in complex LLM inference pipelines. For this purpose, we also present a case study in a real-world industrial setting. The demonstration of the LLM evaluation tool development would help researchers and practitioners in building on-premise systems for LLM evaluation to ensure privacy, reliability, robustness, and reproducibility.

1 Introduction

LLMs have drawn lots of attention recently in both academia and industries (Bang et al., 2023; Zhao et al., 2023). This has led to rapid advancement in building LLM-based applications to solve real-world problems (Fu et al., 2024; Laskar et al., 2023b). However, deploying LLMs in the real world is not trivial. In real-world industrial scenarios, LLMs are required to go through extensive evaluations across benchmark datasets and tasks (Chang et al., 2024; Biderman et al., 2024). Thus, it is crucial not only to achieve high accuracy but also to enhance runtime speed, maintain low memory usage, and protect customer privacy to minimize production costs. Additionally, due to the open-ended nature of responses generated by LLMs, parsing is often needed to evaluate these responses using various metrics (Laskar et al., 2023a, 2024a). These evaluations should be extensible to ensure fair evaluation and reproducibility, especially since diverse teams may contribute to LLM development in industrial contexts. Therefore, consolidating these requirements into a comprehensive evaluation platform is a challenging but necessary task when building LLM-based features in large organizations.

While many frameworks already help address several portions of the LLM evaluation workflow, such as the HELM[1] project, the Big-Bench initiative (Srivastava et al., 2022; Suzgun et al., 2022), LM Evaluation Harness (Biderman et al., 2024), OpenAI evals[2], OpenICL (Wu et al., 2023), and LLMeBench (Dalvi et al., 2023); an industry-standard on-premise evaluation tool that addresses all the requirements mentioned above is still missing. In an industrial context, there may be a large team consisting of scientists, software engineers, and product managers, who may work semi-autonomously on various projects utilizing LLMs. On several occasions, they may need to compare multiple commercially available LLMs (both open-source and closed-source), as well as internally fine-tuned LLMs on different tasks and metrics. In such scenarios, evaluation tools should ensure ease of usage, especially for users who do not have extensive technical expertise (e.g., coding or machine learning knowledge). As many projects may span multiple months, it is required to ensure that these comparisons work with new releases and updates to the models, datasets, and other accompanying artifacts. This poses challenges to the reproducibility

[1] https://crfm.stanford.edu/helm/
[2] https://github.com/openai/evals

Proceedings of the 31st International Conference on Computational Linguistics: Industry Track, pages 286–294
January 19–24, 2025.

and repeatability of the evaluation process with a heterogeneous set of resources. Meanwhile, many of the existing evaluation tools are required to be used via leveraging API endpoints. Therefore, it may not be possible to use these endpoints to evaluate LLMs on sensitive in-house data.

To address these challenges, we have developed a low-code on-premise LLM evaluation tool to help team members reuse large portions of a standardized boilerplate with a seamless implementation to speed up their workflows while also adhering to a specification that is common for the team and reproducible by any member. This tool provides key features required (but missing from existing tools) for evaluating LLM-powered applications in the real world such as multi-query prompt (Laskar et al., 2024b) support, customizable parsing, and other runtime-specific metrics. In this paper, we demonstrate how we build an LLM evaluation tool for real-world industrial scenarios such that practitioners across diverse industries can easily develop similar tools to conduct a comprehensive evaluation of LLMs by ensuring reliability, reproducibility, and privacy before their targeted deployment. In the following section, we first present a scenario in the real-world industrial context to build a product feature powered by LLMs. We discuss the limitations of existing LLM evaluation tools while using them to evaluate various components of the product feature and the importance of building an on-premise system for LLM evaluation. Our proposed industry-focused LLM evaluation tool, **`LLM Evaluate`**, is made publicly available at https://github.com/talkiq/llm-evaluate.

2 Case Study: Evaluating a Real-World Industrial Feature Powered by LLMs

The main objective of this paper is to demonstrate the development of an industry-focused LLM evaluation tool that addresses the challenges that are posed in real-world scenarios while releasing product features powered by LLMs. In this regard, we present a real-world industrial scenario to build an LLM-powered feature for a contact center for certain use cases to ground the evaluation considerations and requirements. More specifically, we study the development of an LLM-powered feature named AiRecaps for a contact center which essentially is an amalgamation of 4 separate tasks:

- **Summarization:** It is a feature that provides a summary of the transcript generated for calls.
- **Action Items:** This feature provides a list of delegated items for further follow-up post call.
- **Call Purpose Categorization:** This feature provides a top-level category for a call based on the main purpose of the call for easier disambiguation and analytics.
- **Call Disposition Categorization:** It is a feature that provides a top-level disposition label on the outcome of the call.

Each of these tasks takes the entire transcript of a call as context.

Requirements: Since the objectives for different tasks in the AiRecaps feature differ significantly, there are variations in terms of prompting, response format, evaluation metrics, etc. For instance, both single-purpose and multi-purpose prompts (Schulhoff et al., 2024; Laskar et al., 2024b) are used, and the output format specified by the prompts could vary from plain text to structured formats such as JSON, YAML, or any other arbitrary format. Furthermore, the evaluation platform should also support a variety of model frameworks. For instance, it is necessary to ensure access to externally hosted APIs, locally loaded models in the major ML frameworks such as Pytorch and Tensorflow, and other optimized model frameworks such as llama.cpp[3], TensorRT-LLM[4], or VLLM[5].

Since the AiRecaps feature that we are studying is geared toward production use, the ability to measure compute costs, and runtime latency, alongside benchmarking performance across heterogeneous hardware platforms while maintaining user data privacy is also necessary. In addition to traditional evaluation metrics, it is important to address the need for adding custom metrics to evaluate the models on proprietary datasets. For instance, measuring the quality of the output from a business context such as toxicity measurements and other generation quality measures (e.g., readability, repetitions, etc.) are required to ensure user satisfaction. Given these concerns, the following set of requirements are needed to be fulfilled to evaluate AiRecaps:

- Evaluate both open and closed-source models.
- Support both public and proprietary datasets.
- Support for multi-purpose prompts.
- Measure runtime performance such as peak memory usage, number of generated tokens, per

[3] https://github.com/ggerganov/llama.cpp
[4] https://github.com/NVIDIA/TensorRT-LLM
[5] https://github.com/vllm-project/vllm

	Custom Datasets	Custom Models	API Models	Multi-query prompts	Custom Parsing	Custom Metrics	Runtime statistics
LLM-Eval	✗	✓	✓	✗	✗	✗	✗
Prometheus-Eval	✗	✓*	✓	✗	✗	✓*	✗
BenchLLM	✓	✗	✓	✗	✗	✗	✗
LLMeBench	✓	✓*	✓	✗	✗	✗	✗
DeepEval	✓	✓	✓	✗	✗	✓	✗
Opencompass	✓	✓	✓	✗	✗	✗	✗
LM Evaluation Harness	✓	✓	✓	✗	✗	✓	✗
LLM Evaluation Tool (ours)	✓	✓	✓	✓	✓	✓	✓

Table 1: Feature coverage of some popular, publicly available LLM evaluation platforms.

token latency, and overall latency.

- Support for independently defining customizable metrics for real-world usage, such as assessments of text generation quality.

- Support the parsing of outputs from different format responses (e.g., JSON or YAML).

Limitations in Existing Tools: Given such a diverse nature of requirements, existing evaluation frameworks fell short in terms of feature coverage. Some of the recent and popular LLM evaluation frameworks are reviewed and their feature coverage is demonstrated in Table 1. Based on our survey, we find several limitations in existing tools that prevent the evaluation of AiRecaps tasks in these tools. For instance, most of these existing tools are aimed towards general-purpose evaluation of LLMs, limited mostly within the academic setting. They featured pre-built blueprints to evaluate against publicly available datasets and tasks, using evaluation metrics that are not applicable in business contexts, and the supports are mostly limited to API-based or public models. Many of these tools also do not support the ability to add new metrics, datasets, or models, while some tools only have limited capability to support a feature (this has been denoted using * in Table 1). While some tools like LLMeBench (Dalvi et al., 2023), OpenCompass (Contributors, 2023), and LM Evaluation Harness (Biderman et al., 2024) come with diverse features, the following issues limit their utilization to evaluate AiRecaps tasks:

Restrictions to a set of models: Support limited to only API-based LLMs (e.g., OpenAI models (OpenAI, 2023), Google's Vertex Models (Team et al., 2023), Claude[6], etc.), or certain open-sourced LLMs (e.g., LLaMA-2 (Touvron et al., 2023), Mistral (Jiang et al., 2023), etc.)

Lacking support for optimized models: The evaluation of optimized models (e.g., GPTQ (Frantar et al., 2022), llama.cpp, Medusa-LLM (Cai et al., 2024), etc.), which is important in real-world scenarios is mostly missing.

Limited to only accuracy-based evaluation: Missing statistics on GPU usage, alongside extensive runtime latency measurement.

Parsing scripts lack generalizability: LLM output parsing scripts are not applicable across responses generated by different LLMs in different task-specific settings (e.g., multi-purpose prompts).

These features are largely missing in all tools explored, which are often required for releasing a high-quality LLM-powered product. To address these concerns, we propose an industry-standard LLM evaluation tool, which we demonstrate in the following section.

3 System Details

The primary goal of the tool is to assist scientists in speeding up evaluation workflows while building LLM-powered features, to ensure reliability in evaluation, reproducibility in the experimental results, and maintenance of privacy. When broken down, the key features this tool supports are:

- **Reliability:** Support a wide range of LLMs, both closed-source and open-source, as well as internally trained, facilitating comparative analysis across different combinations of models, datasets, and/or prompts.
- **Privacy Preservation:** Evaluate internally trained LLMs on proprietary datasets.
- **Compatibility:** Compatible with the commonly used industry standard LLM frameworks (e.g., Pytorch, HuggingFace, llama.cpp, etc). For instance, in addition to the boilerplate created specifically for the tool, it supports (i) a pythonic interface that is compatible with HuggingFace transformers for most open-source LLMs, (ii) the HuggingFace evaluate[7] package for evaluation, (iii) HuggingFace

[6] https://www.anthropic.com/news/claude-3-family

[7] https://github.com/huggingface/evaluate

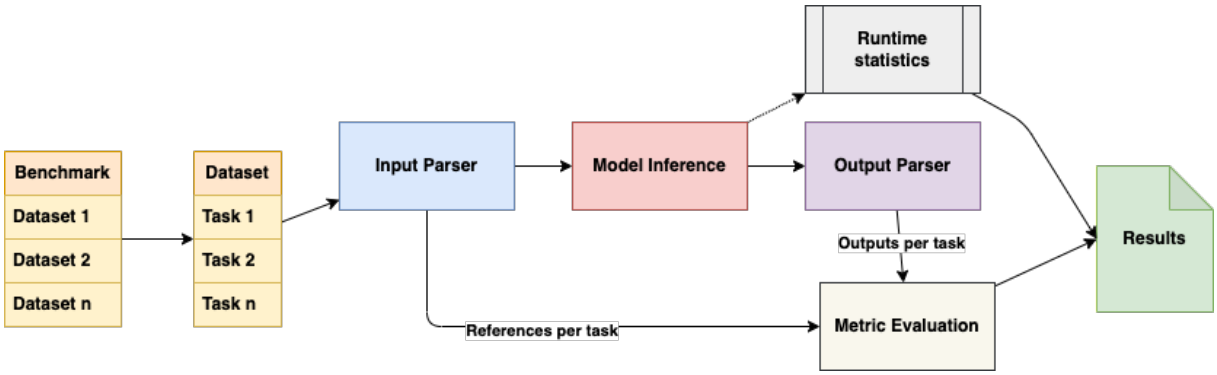

Figure 1: High level overview of the LLM Evaluation Tool

datasets for external, API-hosted datasets. More specifically, this tool interfaces these libraries in a unified manner, enabling a hands-free evaluation environment.

- **Flexibility:** Re-use existing prompt templates and parsing scripts while allowing easy modifications of them. Re-use existing reliable parsing scripts that can also be modified.
- **Robustness:** Measure accuracy alongside runtime latency and memory usage.
- **Reproducibility:** Perform repeatable and reproducible evaluations.

A high-level overview of the proposed LLM evaluation tool is shown in Figure 1.

3.1 Key Components

The tool can essentially be divided into the following five components.

Dataset: A dataset is essentially an encapsulation of the input data that needs to be provided to the model for evaluation. A dataset can be defined by key information such as its source, format (e.g., CSV/JSON/etc.), and columns (input/output) to load and process the data. A key distinction to note here is that a single sample in a dataset can comprise one or more tasks.

Task: A task is the actual objective that is to be addressed by the model. A dataset can contain one or more tasks, which provides metadata necessary for correctly processing individual task-related information for evaluation such as parsing.

Metric: A metric defines the measure to use for evaluation of a model's output given a task. A metric is defined per task per dataset. It is possible to define multiple metrics for a given task.

Parser: A parser (Laskar et al., 2024a) is an intermediary processing layer that can be utilized for both pre and post-model inference text processing. The idea behind this layer is that it allows the extraction of the target output from the descriptive texts to a form that can be easily used to apply various evaluation metrics.

Model: A model is the LLM that will be considered for evaluation. This model can be API based, open-source, or an in-house model. In this regard, a pythonic interface is defined to interact with the model's interface.

Benchmark: A benchmark is a template of a set of tasks that should be performed for an evaluation run. Basically, it comprises a list of datasets, parsers, and metrics that define how we should evaluate the performance of an LLM in various datasets. The idea behind a benchmark is that a group of datasets can be grouped together to form a benchmark that can be independently invoked to create a more semantic starting point for evaluation.

3.2 Initial Configuration

As described previously, the building blocks of the tool are artifacts such as models, parsers, metrics, datasets, and benchmarks. Out of the box, the tool includes Python codes for loading *models* on many popular frameworks such as PyTorch, TensorFlow, HuggingFace Transformers, and HTTP API models. Programming scripts related to commonly used *parsers* and *metrics* are also included. Loaders for *datasets* from cloud storage or HTTP APIs for different *tasks* are also packaged in the tool. To allow team members from various backgrounds to use the tool in a completely no-code environment, only the modifications of configuration files defined in YAML format are enough. Once the necessary blueprint is available, these files can define options such as the configuration necessary to load a model. For instance, a model configuration

(see Table 2) defines the framework of a model, e.g., *HuggingFace transformers* (Wolf et al., 2020), and its more specific type, e.g., *LLaMA-2-7B* (Touvron et al., 2023) model, any loading options, e.g., loading data type, and or any inference options, e.g., sampling parameters during text generation. This negates the need for redefinition of any code artifacts if the tool needs to evaluate any model leveraging the same model blueprint, allowing for quicker integration and usage. Similar YAML design is also applied to the benchmarks (see Table 3) and the datasets (see Table 4).

3.3 Evaluation options

The tool allows for two types of evaluation:

Metric-based Evaluation: Here, metric-based evaluation refers to using metrics like *Precision*, *Recall*, *F1*, *Rouge* (Lin, 2004), etc. for evaluation.

Latency & Memory Usage Evaluation: Memory usage is measured during inference by observing memory usage on NVIDIA GPUs during the evaluation and latency (supports both CPU-only environment and NVIDIA GPUs) is measured by looking at the absolute time taken to produce a complete response given an input prompt.

3.4 Invoking the Tool

Every component in the tool has a blueprint that allows it to load the necessary artifacts correctly. These blueprints map to a Python class in the tool that defines the contract on its usage. For instance, a model class defines a model that can be loaded and the functionality it needs to support to ensure integration with the tool. This model class can be extended to support any segment of models, such as those from the HuggingFace Transformers library (Wolf et al., 2020). Once this specification is created in code, any LLM that uses the HuggingFace Transformer (or other libraries like llama.cpp) can be used with the tool using a no-code approach. This tool can be invoked via a command-line interface. A few options need to be specified when invoking the tool (see Command 1). For instance, for metric-based evaluation (see Command 2):

(a) The benchmark name(s) to load the correct datasets, metrics, and parsers.

(b) The model name or path, which allows for selecting the correct LLM for inference.

Upon receiving these parameters, the following steps happen in sequence:

(a) Load the configuration files.

(b) Load the model.

```yaml
model:
  model: meta-llama/Meta-Llama-3.1-8B
  model_type: hf-automodel
  tokenizer_args:
    model_max_length: 3000
    truncation_side: right
    truncation: longest_first
  model_load_args:
    max_input_tokens: 3000
  model_inference_args:
    max_new_tokens: 512
    num_beams: 1
    temperature: 0.8
  add_to_prompt_start: '[Prompt]'
  add_to_prompt_end: '[Response]'
```

Table 2: Model configuration - contains information such as the model's framework, along with options necessary for model initialization & inference, and appending arbitrary text to every input during inference.

(c) Identify the datasets to be evaluated via the benchmark(s) specified.

(d) For every dataset, (i) load the data, (ii) perform any preprocessing necessary using input parsers (e.g., processing the prompt), (iii) run inference on the model using the processed prompts, (iv) generate the outputs and perform any post-processing using output parsers.

(e) For every task, (i) compute the metric value(s) using the references, and (ii) display the results as a report (see Command 4) and save them with the outputs in the disk.

For latency and memory usage evaluation (see Command 3), the process is very similar to the above steps with the exception that instead of metric-based evaluation, the memory usage and latency are measured by recording GPU memory usage and the wall-clock times. A report is also generated (see Command 5) once the evaluation is complete.

4 Advantage of the proposed LLM Evaluation Tool

As mentioned previously, a key objective of this tool is to ensure flexible, fair, reproducible evaluations of LLMs in real world settings. Thus, this tool needs to be easily extensible to add new models, datasets, and any other processing artifacts, alongside preserving privacy. More specifically:

Flexibility: It ensures a no-code approach to add components that follow an existing blueprint. New models, datasets, and benchmarks can be added by modifying only the YAML configuration.

Compitability and Reproducibility: Configu-

```yaml
benchmarks:
  my_benchmark:
    my_dataset:
      tasks:
        my_dataset_task_0:
          metrics:
          - accuracy
          - f1
          - precision
          - recall
        my_dataset_task_1:
          - rouge
          - MyCustomMetric
```

Table 3: Benchmark configuration - contains a list of datasets and tasks with a list of metrics per task.

```yaml
datasets:
  my_dataset:
    tasks:
      my_dataset_task_0:
        task_type: classification
        key: Task-0
        model_output_parser: AParser
      my_dataset_task_1:
        task_type: generation
        key: Task-1
  column_input: prompt
  column_reference: response
  description: My dataset
  reference_split_parser: AnotherParser
  metadata:
    format: csv
    version: December 11, 2024
    source: gcs
    path: gs://path/to/my/dataset.csv
```

Table 4: Dataset configuration - contains metadata for loading the dataset, a list of task definitions & information about output parsing.

Input Prompt
Provide responses to the following questions in JSON with the key as the question number for the provided context. I called to check on the status of my order. Can you please let me know about it? Task-0: Classify the statement as either positive, negative or neutral. Task-1: Provide a short summary of the passage.
Expected output
{"Task-0": "neutral", "Task-1": "A person called to check on the status of their order."}

Table 5: An example of a multi-query prompt along with its expected output. The evaluation tool can interpret such prompts and also reliably evaluate such outputs.

rations are shareable and reusable between runs, allowing for greater transparency and reproducibility. Resulting reports and output files have the configuration embedded within to allow for disambiguation between evaluation results.

Reliability and Extensibility: Component blueprints can be easily extended in a low-code environment to incorporate new blueprints. For example, a new parser for processing model outputs can be added, or a custom metric for evaluation can be created. This also ensures reliable evaluation for specific business use cases.

Robustness: Allows the evaluation of both metric-based performance and runtime statistics (i.e., memory usage and latency), optimized LLMs (e.g., llama.cpp). In addition, it supports processing and parsing of multi-purpose/multi-query (Laskar et al., 2024b) prompts (as shown in Table 5).

Privacy Preservation: No need to use the tool via public APIs. Thus, no risk of privacy concerns when evaluating models on customer data.

Parallelism: We implement data parallelism so that multiple model instances can be loaded into multiple GPUs to significantly speed up inference through the benchmark. This supports different types of model implementations, such as PyTorch, llama.cpp, Tensorflow, etc.

5 Conclusion

In this paper, we demonstrate an industry-tailored LLM evaluation tool, which ensures flexibility, reliability, privacy preservation, and reproducibility in the evaluation of LLMs in real-world scenarios. No-code environment makes the utilization of this tool straightforward for people with no coding background. We believe that this paper will provide necessary insights on building an easy-to-use evaluation tool for real-world industrial usage to ensure a fair and reproducible evaluation of LLMs. Alongside providing some sample commands to use the proposed LLM evaluation tool (see Appendix A), it has been open-sourced and currently available at https://github.com/talkiq/llm-evaluate.

Limitations

The tool loads evaluation data directly in CPU memory and, therefore, if the evaluation dataset is extremely large, it will need to be partitioned into smaller chunks manually by the end users for processing. GPU memory statistics are only available for NVIDIA's hardware accelerators and CPU memory usage is not reported. Since the focus is to provide support for models fine-tuned on in-house datasets for business-oriented tasks, we do not provide a boilerplate for in-context learning.

Ethics and Broader Impact Statement

This tool is intended for real-world industrial scenarios to ensure robustness, reliability, flexibility, and reproducibility in LLM evaluation. Therefore, it does not pose any ethical concerns.

References

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. *Preprint*, arXiv:2302.04023.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, et al. 2024. Lessons from the trenches on reproducible evaluation of language models. *Preprint*, arXiv:2405.14782.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Fahim Dalvi, Maram Hasanain, Sabri Boughorbel, Basel Mousi, Samir Abdaljalil, Nizi Nazar, Ahmed Abdelali, Shammur Absar Chowdhury, Hamdy Mubarak, Ahmed Ali, et al. 2023. Llmebench: A flexible framework for accelerating llms benchmarking. *arXiv preprint arXiv:2308.04945*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Xue-Yong Fu, Md Tahmid Rahman Laskar, Elena Khasanova, Cheng Chen, and Shashi Tn. 2024. Tiny titans: Can smaller large language models punch above their weight in the real world for meeting summarization? In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 387–394, Mexico City, Mexico. Association for Computational Linguistics.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Md Tahmid Rahman Laskar, Sawsan Alqahtani, M Saiful Bari, Mizanur Rahman, Mohammad Abdullah Matin Khan, Haidar Khan, Israt Jahan, Amran Bhuiyan, Chee Wei Tan, Md Rizwan Parvez, et al. 2024a. A systematic survey and critical review on evaluating large language models: Challenges, limitations, and recommendations. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13785–13816.

Md Tahmid Rahman Laskar, M Saiful Bari, Mizanur Rahman, Md Amran Hossen Bhuiyan, Shafiq Joty, and Jimmy Huang. 2023a. A systematic study and comprehensive evaluation of ChatGPT on benchmark datasets. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 431–469, Toronto, Canada. Association for Computational Linguistics.

Md Tahmid Rahman Laskar, Xue-Yong Fu, Cheng Chen, and Shashi Bhushan TN. 2023b. Building real-world meeting summarization systems using large language models: A practical perspective. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 343–352, Singapore. Association for Computational Linguistics.

Md Tahmid Rahman Laskar, Elena Khasanova, Xue-Yong Fu, Cheng Chen, and Shashi Bhushan Tn. 2024b. Query-OPT: Optimizing inference of large language models via multi-query instructions in meeting summarization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1140–1151, Miami, Florida, US. Association for Computational Linguistics.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. 2024. The prompt report: A systematic survey of prompting techniques. *arXiv preprint arXiv:2406.06608*.

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. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 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.

Zhenyu Wu, YaoXiang Wang, Jiacheng Ye, Jiangtao Feng, Jingjing Xu, Yu Qiao, and Zhiyong Wu. 2023. Openicl: An open-source framework for in-context learning. *arXiv preprint arXiv:2303.02913*.

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*.

A Appendix

```
% llm-evaluate --help

 Usage: llm-evaluate [OPTIONS]
        COMMAND [ARGS]...

 Options
   --help     Show this message and  exit.

  Commands
    benchmark      Run a model against
                   predefined  benchmarks.
    stats-runtime  Get stats on
                   model's runtime.
```

Command 1: Available commands in the proposed tool; metrics-based evaluation via *benchmark* and runtime latency metrics via *stats-runtime*.

```
% llm-evaluate benchmark --help

 Usage: llm-evaluate benchmark
        [OPTIONS] PROFILE

 Run a model against pre-defined benchmarks.

 Arguments
 * profile  TEXT  Path to YAML  configuration
                  profile for the model.
                  [default: None]
                  [required]

 Options
 --benchmarks  TEXT  Optionally specify only a
                     few select benchmarks.
                     e.g. --benchmark demo
```

Command 2: Benchmarking command - runs the metrics-based evaluation. It requires a YAML file containing the necessary configuration. Some optional arguments have been omitted for brevity.

```
% llm-evaluate stats-runtime --help
 Usage: llm-evaluate stats-runtime
        [OPTIONS] PROFILE

 Get stats on model's runtime.

 Arguments
 * profile  TEXT  Path to YAML  configuration
                  profile for the model.
                  [default: None]
                  [required]
```

Command 3: Runtime statistics command - runs the runtime analysis of the model. It requires a YAML file containing the necessary configuration. Some optional arguments have been omitted for brevity.

```
% llm-evaluate benchmark \
    --benchmarks my-benchmark \
    model-profile.yaml

my_dataset@some-version:
  my_dataset_task_0:
    accuracy: 12.0
    f1-macro: 13.0
    f1-micro: 14.0
    f1-weighted: 15.0
    precision-macro: 16.0
    precision-micro: 17.0
    precision-weighted: 18.0
    recall-macro: 19.0
    recall-micro: 19.0
    recall-weighted: 18.0
  my_dataset_task_1:
    rouge1: 11.0
    rouge2: 12.0
    rougeL: 13.0
    my-custom-metric: 99.0
```

Command 4: Sample output for the *benchmark* command. It contains performance-based metrics as defined in the configuration for the model over the selected benchmark's datasets.

```
% llm-evaluate stats-runtime model-profile.yaml

Report
------
gpu_memory:
  mean: 9.089
  p01: 7.006
  p95: 9.434
  p99: 9.434
  peak: 9.434
  stdev: 0.762
inference_stats:
  input_tokens:
    mean: 842.095
    p01: 1.0
    p95: 2955.35
    p99: 3268.04
    stdev: 879.697
  latency_s:
    mean: 0.228
    p01: 0.037
    p95: 1.32
    p99: 1.66
    stdev: 0.394
  output_tokens:
    mean: 3.158
    p01: 2.0
    p95: 5.0
    p99: 5.0
    stdev: 1.3
  time_per_token_ms:
    mean: 89.386
    p01: 8.106
    p95: 462.976
    p99: 790.43
    stdev: 165.487
load_stats:
  mean: 1.107
  p01: 0.955
  p95: 1.247
  p99: 1.259
  stdev: 0.155
```

Command 5: Sample output for the runtime statistics command. The output contains runtime statistics for the model such as the GPU peak memory used, model load time, number of input & output tokens, and the inference latency.